UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

20|21

Trabajo fin de grado

Plataforma de Monitorización y Descubrimiento de Vulnerabilidades

Gonzalo Jurado Pallarés

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

excelencia Campus Internacional UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Plataforma de Monitorización y Descubrimiento de Vulnerabilidades

Autor: Gonzalo Jurado Pallarés
Tutor: Álvaro Ortigosa Juárez

junio 2021

Gonzalo Jurado Pallarés
Plataforma de Monitorización y Descubrimiento de Vulnerabilidades

Gonzalo Jurado Pallarés
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia y amigos

Las palabras son de todos, hasta que uno demuestra que es capaz de apropiarse de ellas.

Jöel Dicker -

Prefacio

Éste trabajo de fin de grado ha sido realizado con varios propósitos. El primero de ellos consiste en el aprendizaje de una de las ramas más importantes actualmente dentro de la ingeniería informática a nivel mundial: la ciberseguridad. El objetivo es que tanto por realizar el trabajo como por exponerlo, adquiera unos conocimientos básicos que me permitan a entender algunas de las muchas metodologías que existen dentro de éste ámbito.

Otro propósito es llevar a cabo un proyecto que esté en constante crecimiento, ya sea desde su fase de planteamiento y organización hasta después de la entrega de este propio documento, pues el aprendizaje nunca tiene un techo, siempre se puede extender y mejorar, y el objetivo es que mi trabajo también lo haga.

Por último, espero poder demostrar las capacidades que los estudiantes somos capaces de adquirir y de asimilar a lo largo de un grado como éste, y aplicar todos los conceptos en la medida de lo posible, y como creador de este trabajo ser capaz de exponerlo a cualquier tipo de lector para que aprenda y valore el esfuerzo realizado.

Gonzalo Jurado Pallarés

Agradecimientos

Antes de empezar, me gustaría agradecer a todas las personas que me han acompañado durante estos 4 años de incansable trabajo y esfuerzo.

Sin ninguna duda, a mis padres que me han apoyado en todas las decisiones que he tomado tanto buenas como equivocadas, y han sido capaces de levantarme incluso en mis peores momentos. A mi hermano por ser mi protector y referente, ya que nunca me cansaré de aprender de él. Sin ellos no sería quien soy a día de hoy, y mi mayor objetivo es hacerles sentir orgullosos.

Además, quiero agradecer a mis amigos por hacer el viaje mucho más sencillo y agradable, y a mi novia por ser el pilar que me mantiene en pie día tras día, dándome fuerzas cuando no podía sacarlas de ningún otro sitio, y acompañándome de forma constante con un cariño incondicional.

Por supuesto, agradecer a Álvaro Ortigosa Juárez por ofrecerse a acompañarme en este trabajo y confiar en mi para esta recta final en el camino universitario, y por la ayuda y el seguimiento que ha aportado a lo largo de estos meses.

Resumen

El trabajo de fin de grado que se presenta tiene como finalidad el desarrollo de una herramienta capaz de automatizar el proceso de análisis y monitorización de activos de una organización enfocado a la búsqueda de vulnerabilidades de forma automática, simulando el trabajo que se realiza de forma manual en las metodologías habituales de *pentesting*, orientado a un concepto en auge hoy en día en el área de la ciberseguridad, los Bug Bounties.

Esto será posible mediante la programación de un servidor en Django, que permitirá al usuario registrar escaneos cuyo objetivo sea una organización en particular, y que a partir de un dominio inicial, sea capaz de obtener una cantidad considerable de posibles vulnerabilidades existentes dentro de los servicios web internos, respetando el alcance establecido por la organización.

La metodología para lograr esto será tener módulos que se repartan las principales tareas para obtener información a partir de ese dominio: enumeración de subdominios, escaneo de puertos, descubrimiento de servicios web y descubrimiento de vulnerabilidades. De forma directa el usuario puede configurar un escaneo para llevar a cabo las tareas que desee, pudiendo incluso personalizar las herramientas que quiera utilizar en alguna de ellas.

Mediante una adecuada gestión de los recursos la herramienta desarrollada será capaz de registrar una gran cantidad de escaneos personalizables y lanzarlos de forma paralela con un alto rendimiento, para posteriormente poder encontrar toda la información extraída guardada en una base de datos local y mostrada en un formato sencillo en la interfaz de la herramienta.

La herramienta predominante en el funcionamiento y la eficiencia de este proyecto será Nuclei, que gracias a sus reglas de detección de vulnerabilidades nos permite encontrar algunas realmente importantes, cuyos reportes son valorados en miles de euros, como por ejemplo Server Side Request Forgery, Local File Inclusion / Remote Code Execution o Apache SOLR.

Palabras clave

Seguridad Informática, Escaneo de subdominios, Pentesting, Bug Bounties, Descubrimiento de Activos, Escaneo de Puertos, Django, Análisis y Monitorización, Vulnerabilidades en Servicios Web, Explotación de Vulnerabilidades

Abstract

The purpose of the final degree project presented is the development of a tool capable of automating the process of analysis and monitoring of assets of an organization focused on the automatic research of vulnerabilities, simulating the work that is done manually in the usual methodologies of *pentesting*, oriented to a concept booming today in the area of cybersecurity, the Bug Bounties.

This will be possible by programming a server in Django, which will allow the user to register scans targeting a particular organization, starting from the specified domain. Then it will be able to obtain a considerable amount of possible vulnerabilities existing within the internal web services, suiting the scope established by the organization.

The methodology applied to achieve this has been to have modules that distribute the main tasks to obtain information from that domain: subdomain enumeration, port scanning, web services discovery and vulnerability discovery. The user can directly configure a scan to carry out the tasks he/she wants, and can even customize the tools he/she wants to include in some of them.

Through proper resource management the developed tool will be able to record a large number of customizable scans and launch them in parallel with high performance, and then find all the extracted information stored in a local database and displayed in a simple format in the tool interface.

The predominant tool in the operation and efficiency of this project will be Nuclei, which thanks to its vulnerability detection rules allows us to find some really important vulnerabilities, whose reports are valued in thousands of euros, such as Server Side Request Forgery, Local File Inclusion / Remote Code Execution or Apache SOLR.

Keywords

Computer Security, Subdomain Scanning, Pentesting, Bug Bounties, Asset Discovery, Port Scanning, Django, Analysis and Monitoring, Web Services Vulnerabilities, Vulnerability Exploiting

Índice

Listas

Lista de códigos

Lista de figuras

1

Introducción

El uso de las redes informáticas se ha vuelto esencial dentro del progreso evolutivo del ser humano en la época contemporánea. Es por ello que las organizaciones más grandes del mundo son incapaces de persistir sin exponer sus servicios de alguna forma virtual, ya sea a través de comercios, infraestructura o sencillamente para servirse de la Web para hacerse conocer por muchas más personas a nivel global. Sin embargo, no es oro todo lo que reluce, y en las profundidades de la red existen fines que no conocen de bondad o legalidad, y que pueden resultar aterradores para los usuarios de a pie. Exposición de información sensible, filtración de credenciales, datos personales, y un sin fin de trágicos acontecimientos que pueden llevar a la ruina hasta a las más grandes empresas.
A raíz de la creación de infraestructuras informáticas, nacen las necesidades de protegerlas, y ser capaces de reconocer la brecha —la cual por pequeña que sea, siempre existe— en el muro de titanio. Es ahí donde entra en juego el papel de la ciberseguridad.
Este documento servirá como guía para introducirse a una de las muchas áreas que se hallan dentro del campo de la ciberseguridad, como es la detección de vulnerabilidades en servicios Web. Se reconocerá el procedimiento básico principal que seguiría un auditor a la hora de realizar un ejercicio de intrusión supervisado por una organización, y se aprenderá a diferenciar las distintas etapas dentro del proceso de trabajo para obtener los resultados deseados.

1.1. Motivaciones

El origen de la idea para este trabajo de fin de grado nace en el concepto en auge de los **Bug Bounties**. Dicho término puede resultar poco familiar, ya que se asocia a una nueva mecánica que se ha vuelto muy importante dentro del mundo de la ciberseguridad en los últimos años. Se trata, a grandes rasgos, de programas de recompensa públicos, establecidos por organizaciones de cara a que cualquiera tenga la oportunidad de buscar fallos dentro de los dominios que la propia organización facilite. ¿Por qué en particular los programas de *Bug Bounty*, y no otro tipo de procesos de auditoría mas reconocidos como ejercicios de *Pentesting* o de *Red Team*? [1] Se trata de una aproximación menos técnica de todas las que se podrían plantear, y servirá para elaborar un proceso de aprendizaje más sencillo, teniendo en cuenta el tiempo disponible para desarrollar este trabajo.

Estos programas de recompensa no entran en términos tan específicos dentro de la ciberseguridad como los otros comentados. Sin embargo, abarcan una serie de pasos suficientemente importantes como para adquirir un punto de vista global sobre todos los conceptos en un periodo breve de tiempo. La teoría es fácil de entender: se obtiene el alcance —en ocasiones, se especifica como 'scope'— que la organización haya proporcionado y se intenta adentrar en lo más profundo de su sistema para poder encontrar algún fallo y reportarlo. Sin embargo, la práctica es menos intuitiva. El principal foco de motivación que obtuve tras pensar en dicha idea era el hecho de aprender cosas en un ámbito prácticamente desconocido como es el de la seguridad informática. A lo largo del grado que he estudiado, uno conoce con brevedad ciertos temas o incluso repasa algunos aspectos interesantes, sin embargo nunca se adentra en profundidad en dicha rama. Por todo esto, sentí cierta atracción a aprender la metodología que se sigue, e intentar replicarla de la mejor forma posible y en el mejor de los casos, obtener algún resultado prometedor. Se considerará resultado prometedor el hecho de poder encontrar vulnerabilidades que, a pesar de probablemente haber sido descubiertas por mucha otra gente por ser programas públicos, supongan riesgos reales existentes dentro de la organización, independientemente de su criticidad.

El primer paso estaba definido, ya había una idea moldeándose. Sin embargo, tras pensar en ello, me di cuenta de que realmente mi idea no hacía uso de habilidades que ya hubiese aprendido a lo largo de la carrera, y que sin duda me podrían ser muy útiles para un mejor desarrollo. Entonces, decidí dar el segundo paso: juntar las piezas del puzle para idear algo que por partes iguales me sirviese para aprender cosas nuevas, además de aprovechar los conocimientos ya adquiridos. La conclusión fue clara: programar una plataforma que sirviese para automatizar la realización de las diferentes etapas —las cuales detallaré posteriormente— que involucran el proceso de encontrar vulnerabilidades, y ser capaz de cohesionarlas de una forma rapada y eficiente, que le permita al auditor realizar múltiples escaneos de distintos tipos de manera simultánea, ahorrando así mucho tiempo. A partir de ahí, he llevado a cabo un proceso de profundización de los estudios, el cual detallaré a continuación, para entender con exactitud algunas de las etapas que tendría que definir dentro de los objetivos para mi —ya oficial— herramienta. A continuación, definiré de forma clara y concisa los objetivos establecidos para este trabajo y el resto del documento.

1.2. Objetivos

En este apartado quedarán definidos los objetivos para el proyecto y el resto del documento, desde distintos puntos de vista. Por un lado, voy a establecer unos objetivos de carácter personal, que sirvan de referencia para entender lo que quiero conseguir mas allá de lo práctico o didáctico. Además, se fijarán también unos objetivos de aprendizaje para poder garantizar que se obtiene alguna ganancia adicional, mas allá del propio resultado tangible a través del código o la plataforma construida. Por último, pero no menos importante, se incluirán unos objetivos de desarrollo, donde quede en constancia

lo que se quiere lograr a través de la herramienta que se va a implementar, los requisitos que deberá cumplir y la funcionalidad que se desea que quede cubierta tras su finalización pertinente.

1.2.1. Objetivos personales

Para empezar, uno de los objetivos principales será el hecho de ser capaz de trabajar de forma individual en un trabajo completo y relativamente complejo que requiera de forma equilibrada un esfuerzo en desarrollo y en estudio, que permita aprender conceptos nuevos, así como aplicar los ya adquiridos. En segundo lugar, ser capaz de llevar a cabo una organización personal que permita hacer un seguimiento del trabajo semanalmente y evidenciar el progreso de forma tangible, estableciendo sub-objetivos temporales. El último, y quizá el mas importante de todos, será el hecho de terminar satisfecho con los resultados obtenidos y con el crecimiento personal a lo largo de la realización de este trabajo de fin de grado.

1.2.2. Objetivos de aprendizaje

De cara al aprendizaje, se asegurará el aprendizaje de los conceptos esenciales de las principales etapas que se encuentran dentro de un proceso simplificado de auditoría y de qué tipo de actividades se componen dichas etapas, como por ejemplo:

- **O-1.–** Descubrimiento de subdominios.
  - **O-1.1.–** Herramientas
  - **O-1.2.–** Motores de búsqueda
- **O-2.–** Enumeración de servicios
  - **O-2.1.–** Escaneo de puertos
  - **O-2.2.–** Búsqueda de hosts
- **O-3.–** Descubrimiento de vulnerabilidades
  - **O-3.1.–** Principales tipos de vulnerabilidades
  - **O-3.2.–** Reglas de reconocimiento
  - **O-3.3.–** Herramientas
  - **O-3.4.–** Detección y explotación

Algunos de estos contenidos serán aplicados solo desde el punto de vista teórico, como por ejemplo, la explotación de vulnerabilidades. De cara al aprendizaje más específico dentro del ámbito de la ciberseguridad, un objetivo adicional de cierto interés será el hecho de brindar la posibilidad de aprender e implementar algunas reglas de detección personalizadas para poder incorporarlas posteriormente en los escaneos dentro de la propia herramienta. El concepto de regla de detección quedará mejor definido cuando se llegue a la etapa de especificación del tipo de herramientas externas que se van a incorporar.

1.2.3. Objetivos técnicos

El desarrollo tendrá como principal meta implementar una herramienta que sea capaz de automatizar las distintas etapas comentadas anteriormente, y ser capaz de configurar los distintos procesos de búsqueda al alcance de un click. El sistema tendrá un soporte de datos persistente, que de forma ideal será capaz de actualizarse de forma periódica a partir de programación de actividades, sin la necesidad de que el usuario esté presente. La herramienta mostrará toda la información obtenida de una forma clara y directa al usuario, y le permitirá navegar a través de ella de una forma eficiente para poder encontrar los datos que más le interesen.
Por lo tanto, la funcionalidad más relevante que se podrá llevar a cabo con la herramienta será la siguiente:

O-1.– El usuario podrá añadir escaneos, aportando un nombre de organización y los dominios de los que quiera partir.

O-2.– El usuario podrá editar las distintas configuraciones de escaneos, de cara a personalizar la búsqueda de información.

O-3.– El usuario podrá mantener múltiples escaneos de forma simultánea ejecutándose, con diferentes configuraciones.

O-4.– Se podrá consultar en todo momento el estado de un escaneo y sus resultados.

Además, habrá ciertos objetivos adicionales que dependerán del grado de progreso que se obtenga a lo largo del desarrollo de la herramienta, pero algunos de ellos son:

O-1.– Posibilidad de configurar las herramientas adicionales que se utilicen para cada tipo de escaneo.

O-2.– Posibilidad de programar escaneos de forma automática con periodos de tiempo establecidos.

O-3.– Posibilidad de añadir reglas propias de detección de vulnerabilidades.

O-4.– Posibilidad de generar reportes automáticos de vulnerabilidades a diferentes plataformas u organizaciones.

Sin embargo, lo más importante y valioso de la herramienta que se va a implementar es el hecho de que su desarrollo se enfocará de cara a conseguir que sea ampliamente extensible, que sea capaz de mantener una progresión incluso después de haber concluido este mismo trabajo de fin de grado, para que ésta siga creciendo e incorporando nuevos tipos de funcionalidades, a medida que se vayan ampliando los conocimientos técnicos dentro del ámbito de la ciberseguridad. Además, se buscará que en todo momento la herramienta tenga un rendimiento óptimo, haciendo un uso eficiente de los recursos e incluso llegando a ampliarlos en caso de ser necesario, en función de la tarea que se tenga que llevar a cabo.

1.3. Alcance

Antes de progresar a los siguientes capítulos de este documento, es importante que quede correctamente definido el alcance del trabajo. Principalmente, debe constar que a día de hoy existen múltiples herramientas que realizan funcionalidades como la que yo busco. Dichas herramientas ya realizan tareas similares como unificar distintos procesos para poder escanear de forma automática subdominios, hosts web o vulnerabilidades. Es por esto que tiene constar que el objetivo principal del proyecto no es otro que de aprender todos estos conceptos de forma específica, ya que de otra forma no sería necesaria su realización. La definición de estas herramientas se ha establecido en el Anexo B por motivos de espacio en el contenido del documento.

Sin embargo, sí se usarán herramientas de terceros para llevar a cabo cada tarea en particular. Es decir, se incorporarán funcionalidades ya existentes, como por ejemplo enumerar subdominios a partir de un dominio; sin embargo, se buscará unificar todos estos procesos de forma efectiva. Éstas se mencionarán dentro del contexto correspondiente en los siguientes capítulos, cuando se aborde la fase de estudio e implementación. Tampoco es un objetivo que la herramienta supere el trabajo de un experto en ciberseguridad, debido a motivos evidentes. Bajo ningún concepto se busca hallar vulnerabilidades que no sean conocidas por otras personas, pero sí será un objetivo que se entiendan y se aprenda a reconocerlas, de nuevo en la misma línea que lo comentado anteriormente. La herramienta debería servir para introducir a cualquier persona iniciándose a los conceptos básicos y facilitar su proceso de aprendizaje, pero también para simplificar las fases iniciales de búsqueda a un profesional con conocimientos avanzados, teniendo él que profundizar de forma manual a partir de ese punto.
Si bien la arquitectura, composición del sistema web o incluso el estilo de la herramienta son importantes, no será el principal foco de atención a lo largo del desarrollo de la plataforma, por lo que, no será importante sacrificar aspectos técnicos dentro de estos ámbitos de cara a obtener un mejor resultado dentro de los objetivos establecidos previamente.
Por ende, el punto de vista es plenamente didáctico; ser capaz de entender cómo funcionan este tipo de herramientas ya existentes, a partir de desarrollar una por mi propia cuenta, que sin embargo aproveche programas sencillos para realizar cada una de las tareas individuales para así no obstaculizar el proceso de aprendizaje obligando a entrar en tareas más complejas.

2

Diseño

Una vez presentados los objetivos, se procederá a describir el diseño propuesto para la herramienta. De cara a ser capaz de abordar bien el trabajo, hay que definir unas pautas muy específicas, desde la arquitectura de sistema que se va a mantener hasta el diseño de las estructuras de datos que se van a utilizar, o incluso las herramientas que se quieran incorporar en la plataforma para hacerlas funcionar de forma simultánea.

Este capítulo va a contener tres apartados principales; por un lado, se detallará el proceso de formación e investigación que se debe llevar a cabo de cara a conocer la metodología que se quiere implementar a nivel de seguridad, conocer las herramientas principales que existen para realizar las tareas que deberá llevar a cabo la plataforma y entender el alcance del desarrollo principal que se quiere cubrir respecto a las amplias posibilidades existentes. Por otro lado, será necesario especificar los requisitos tanto funcionales como no funcionales de la herramienta, ya que esto es esencial en la fase de diseño de cualquier desarrollo software, de cara a tener unos objetivos oficialmente establecidos. Finalmente, se definirá con exactitud la arquitectura del sistema informático que se quiere construir.

2.1. Profundización de los estudios

Se procede a realizar una fase en la que se investigará de cara a entender el contexto teórico de las etapas dentro de un procedimiento de análisis de sistemas, aprender las metodologías existentes y se definirá una estructura que se va a seguir en la implementación de la herramienta. Convencionalmente, los diferentes tipos de análisis ya sean de *red team* o *pentesting*, o meramente de limpieza dentro de una organización siguen los mismos pasos. La diferencia a grandes rasgos es el nivel de conocimiento que tiene la propia organización de la tarea que se está llevando a cabo por el equipo de seguridad. Todas sin embargo comienzan por una fase de reconocimiento, la cual permite analizar el sistema que se está a punto de analizar y conocer la extensión de éste. En muchas ocasiones, en esta misma etapa se llegan a encontrar recursos de los cuales la organización no es conocedora, y en los que muchas veces se hallan muchos vectores de acceso. Es por esto que dicha fase es crucial a la hora de realizar un escaneo, por lo que se realizará una definición exhaustiva.

A través de una búsqueda detallada, se obtiene el siguiente esquema generalizado en la figura 2.1 [2]

que aplica todo lo que se podría desear en una etapa de reconocimiento.

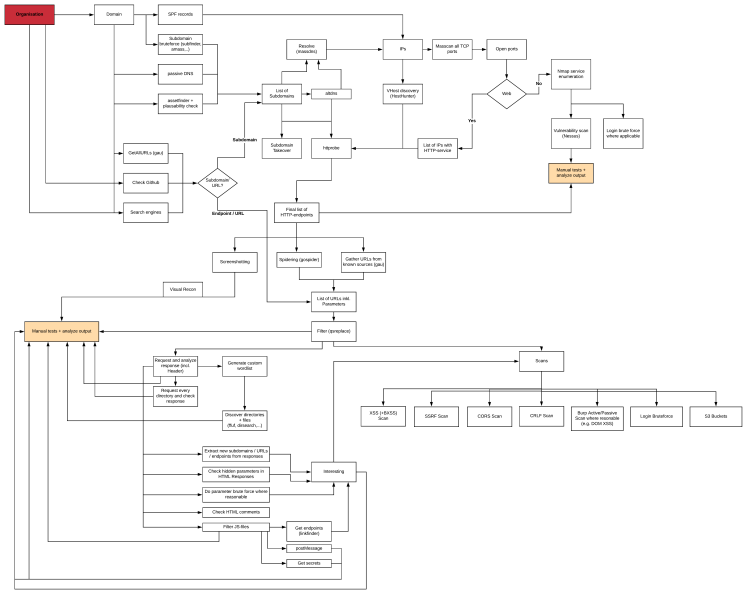


Figura 2.1: Diagrama que refleja las etapas dentro del análisis de un sistema

Como se puede comprobar, el diagrama es muy extenso, ya que refleja en detalle todo el proceso que se tendría que seguir para tener unos resultados óptimos. Sin embargo, como se ha mencionado en el primer capitulo, uno de los objetivos de la plataforma es ser completamente extensible, por lo que el planteamiento inicial será simplificado. Obteniendo como referencia el diagrama superior, a continuación se detallarán las 3 funciones principales dentro de la herramienta.

2.1.1. Enumeración de subdominios

La primera funcionalidad básica que se necesita cubrir es la fase de enumeración de subdominios. Esta etapa es clave ya que de ella dependerá el número de resultados que se obtengan posteriormente, así como la calidad de ellos. [3]
Esta etapa consiste en, a partir de un dominio base —en muchas de las ocasiones se tratará del TLD, pero no lo será necesariamente— obtener una lista de todos los subdominios que éste contenga. Dentro de la enumeración de subdominios, se pueden reconocer dos correlaciones de los dominios: la vertical y la horizontal [1]. Si se imagina una estructura de dominios jerárquica con el TLD como nodo raíz, la correlación vertical consiste en reconocer todos los nodos que sean hijos del nodo raíz. En el caso de la correlación horizontal, estaríamos hablando sobre adquisiciones relacionadas con la entidad de origen. En la figura 2.2 se puede entender de forma gráfica estas definiciones.

Es importante entender que en la herramienta que se va a desarrollar se aplicará la correlación vertical de subdominios, ya que a la hora de tener un alcance definido por una organización, no se podrán analizar las posibles adquisiciones que tenga la propia organización. Además de este punto,

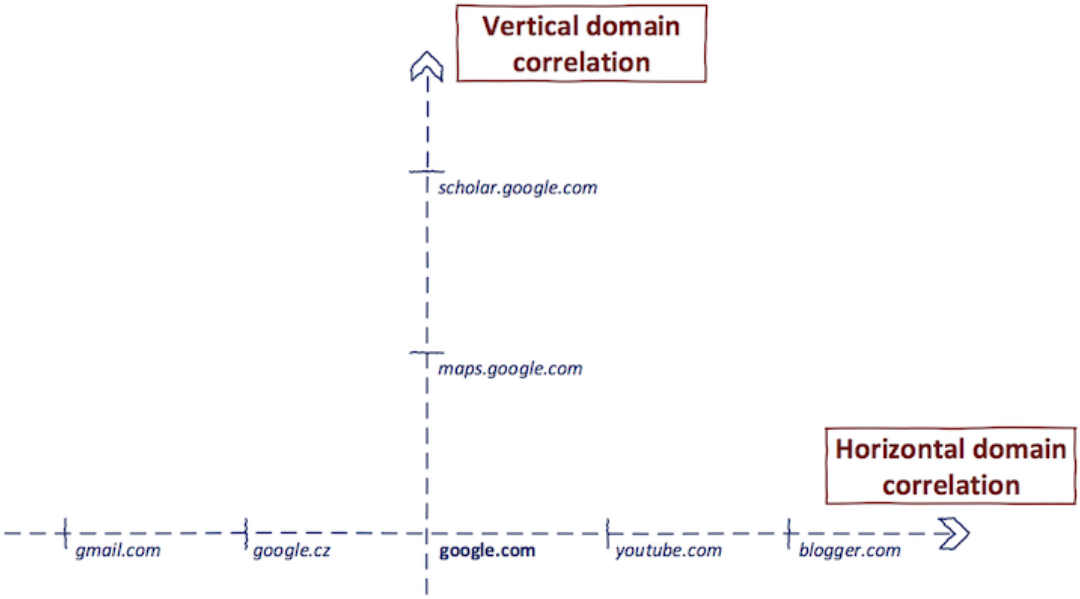


Figura 2.2: Correlación vertical y horizontal para el dominio google.com

es importante definir una buena estructura para compaginar mas de una herramienta que realice la enumeración de subdominios, para obtener resultados desde distintos puntos de enfoque, siempre teniendo en cuenta la necesidad de que esta etapa sea lo más rápida posible, ya que las siguientes que se comentarán a continuación son inevitablemente costosas en tiempo.
Una vez definido el objetivo a la hora de enumerar subdominios, pasaremos a mencionar una serie de herramientas que realizan esta función, que podrán ser interesantes para incorporarlas en la futura herramienta de cara a poder combinarlas para poder obtener una mayor cantidad de resultados.

AssetFinder

AssetFinder es una de las herramientas básicas para la tarea de enumeración de subdominios [2]. Se trata de una herramienta de código abierto que se puede encontrar en github [4], y sencillamente se encarga de enumerar dominios y subdominios potencialmente relacionados con un dominio dado. Una utilidad notable de esta herramienta es la posibilidad de incluir un fichero que contenga más de un dominio, para que se realice la enumeración con todos ellos.

SubFinder

La segunda herramienta de código abierto que se va a incluir es SubFinder [2] [5], cuya funcionalidad esencial es descubrir subdominios válidos para servicios web de forma pasiva. El aspecto pasivo implica que no se realizará ninguna tarea de fuerza bruta, es decir, toda la información obtenida por la herramienta es a través de encontrar dominios sin intervenir activamente o explotar los sistemas.

Amass

La desventaja de las dos herramientas anteriores es que obtienen resultados de una forma más básica. Amass [6] se trata de una de las herramientas por excelencia para la enumeración de subdominios [7]. Principalmente, el tipo de escaneo que realiza es activo, además de que se encarga de utilizar una amplia variedad de fuentes para obtener dominios. Aplica resoluciones DNS, web scraping, uso de certificados o APIs existentes. Junto a todo esto, es capaz de obtener un número considerable de subdominios, con lo cual es una herramienta imprescindible.

Fuerza Bruta

Adicionalmente a estas herramientas, existen técnicas que serán útiles para encontrar dominios adicionales, que no se conozcan únicamente a través de los certificados, buscadores, etc. Una de esas funcionalidades es la inclusión de técnicas de fuerza bruta que sean capaces de encontrar dominios que no sean accesibles desde buscadores o que no tengan certificado mediante técnicas de generación de permutaciones para probar distintas combinaciones.

De forma particular a este trabajo, se van a utilizar el siguiente esquema: la herramienta principal para generar permutaciones es DNSCewl [8] de codingo. Ésta se utiliza sobre la lista de subdominios ya encontrada con las herramientas previas, y con una serie de wordlists prueba combinaciones aleatorias. Esto genera una gran cantidad de dominios, muchos de los cuales probablemente no existen realmente, por lo que se utilizará la herramienta PureDNS [9], la cual obtiene todas las permutaciones realizadas y utiliza MassDNS [10], la herramienta por excelencia de resolución de subdominios, de forma que así se verifique cuales realmente son verdaderos. El motivo de usar PureDNS es que incluye un balanceo de carga para distribuir las peticiones de MassDNS a todos los servidores DNS posibles. Además, también realiza una gestión adicional de wildcards. Además, se utilizan herramientas como Anew [11] o Unfurl [12] para limpiar los posibles resultados generados de duplicados o formatos no válidos. Todo esto se incluirá de cara a maximizar los resultados obtenidos en esta fase, para mejorar los posibles resultados todo lo deseado.

2.1.2. Enumeración de servicios

El segundo paso tras enumerar todos los subdominios de la organización consiste en analizar y enumerar los servicios existentes dentro de ellos, para pasar a escanear las vulnerabilidades. Dentro de la enumeración de servicios predominará de forma inicial el escaneo de servicios web, sin embargo a medida que evolucione la plataforma se irán incorporando herramientas de detección para reconocer otro tipo de servicios dentro de la infraestructura de la organización, y realizar tareas específicas de búsqueda de vulnerabilidades según el tipo de servicio. El motivo de esta división es que el ámbito de los servicios web es predominante y normalmente el principal foco de atención en la búsqueda de

vulnerabilidades.

Descubrimiento de servicios web

Como se ha comentado anteriormente, lo servicios web serán el principal objetivo en las versiones iniciales de la herramienta para abordar la búsqueda de vulnerabilidades, ya que abarcan una gran superficie dentro de dicho ámbito por lo que se podrá realizar un aprendizaje lo suficientemente completo sobre ello. La principal herramienta que se va a utilizar para esta tarea es httpx.
Httpx [13] es una herramienta de código abierto de ProjectDiscovery, los mismos desarrolladores de subfinder y algunas otras que se mencionarán posteriormente, cuyo uso principal es el descubrimiento de hosts web a través del protocolo http y https, dado un dominio de internet. Esto nos permitirá a enlazarla con los resultados previos de la fase de enumeración de subdominios, para obtener de ahí los servicios web. Alguna información de interés que se puede obtener acerca de los hosts gracias a esta herramienta son el puerto en el que se halla el servicio, el código de respuesta de la petición realizada así como el content length. Sin embargo, la información de mayor utilidad será el cname, el tipo y versión de servidor sobre el que se halla el host, y un listado de las tecnologías que utiliza.
Es importante tener en cuenta que toda esta información no será diferencial a la hora de incorporar los programas que analicen las posibles vulnerabilidades, por lo que no será aprovechada de forma directa. Sin embargo, esto no significa que no sea conveniente, ya que son parámetros que se pueden tener en cuenta para hacer auditorías de forma manual. El objetivo será que a medida que el usuario aumente sus conocimientos técnicos de ciberseguridad sea capaz de aprovechar esta información para hacer trabajos manuales que le otorguen unos resultados adicionales provechosos, y éste es el motivo por el que nos interesa esta herramienta.

Escaneo de puertos para otros servicios

De forma paralela a los servicios web, se planteará la posibilidad de que la herramienta evolucione a un rango mayor, siendo capaz de generalizar su función a todo tipo de servicios existentes, ya sean de gestores de bases de datos, protocolos como FTP, SSH y más. Para ello, se hará un análisis adicional sobre los subdominios encontrados, que consistirá en un escaneo de puertos. Dicho escaneo se llevará a cabo con la herramienta Naabu [14] desarrollada por ProjectDiscovery. El resultado será el descubrimiento de puertos adicionales a los utilizados para el protocolo http/s (80 y 443), que nos darán información sobre los servicios adicionales que tenga la organización dentro de su jerarquía de activos. Esto se podrá combinar con otras herramientas que permitan analizar dichos servicios en profundidad, como por ejemplo nmap.
En las versiones iniciales sin embargo, no se entrará en profundidad dentro de éste ámbito, sencillamente se mantendrán los puertos como información adicional para los subdominios, al igual que en el apartado anterior, para tener todos los datos de interés dentro de la herramienta y poder consultarlos y utilizarlos en versiones futuras.

2.1.3. Análisis de vulnerabilidades

La etapa final dentro del análisis que se va a elaborar a través de la herramienta es el descubrimiento de vulnerabilidades. En primera instancia, se analizarán las vulnerabilidades de los hosts web que se hayan descubierto en la fase anterior. Esto evidencia la necesidad de aplicar las distintas etapas en el orden predefinido, de cara a su correcto funcionamiento y resultado. Esta última parte consistirá en reconocer la herramienta que servirá para realizar esta tarea.

Nuclei Templates

Una de las ventajas que tiene esta herramienta es el repositorio de nuclei-templates. Este repositorio contiene una extensa lista de reglas codificadas en yaml que se utilizan para reconocer posibles vulnerabilidades en los diferentes sistemas existentes. Dicha lista además se va actualizando con cierta regularidad, y actualizarla de manera local es tan fácil como ejecutar un simple comando. Esto no solo nos da una amplia posibilidad de detección de un sinfin de fallos diferentes, ya sean errores de configuración, subdomain takeovers, fallos sobre redes de comunicación, cves, etcétera, sino que también brinda la posibilidad de generar con libertad absoluta las reglas que se deseen, siempre y cuando se haga en ese mismo formato. Esto supone una ventaja para aquellos expertos en ciberseguridad que por su experiencia conocen técnicas que no conoce otra gente, y les permite incorporarlas de forma automática dentro de la herramienta.
Uno de los objetivos mencionado correspondientemente en su apartado será el hecho de aprender una o varias reglas de forma técnica y teórica, y a ser posible no existentes dentro de los templates por defecto, de forma que se generen de forma manual las propias reglas en el formato adecuado y se incorporen a la herramienta. La lista de templates es tan amplia, que no se entrará en detalle en esta sección en todas y cada una de las reglas existentes, pero en la fase de pruebas se incluirán los resultados obtenidos y se irán analizando las reglas con mayor severidad que haya sido capaz de detectar la plataforma.

2.2. Requisitos

En este apartado se especifican los requisitos tanto funcionales como no funcionales que se han de establecer en esta fase de diseño. Es necesario fijar estos requisitos en las fases iniciales para poder realizar un seguimiento del progreso que se realiza a lo largo del desarrollo y demás etapas de la creación de la herramienta. Los requisitos funcionales definen las funcionalidades mínimas —esto es importante, ya que en cualquier momento se podrán extender de las aquí establecidas, ya que el objetivo de la herramienta es que sea capaz de seguir una evolución constante, mas allá incluso del propio trabajo de fin de grado— que se desea que se cubran con la aplicación. Por otro lado, los

requisitos no funcionales son aquellos que no se refieren de forma directa a la funcionalidad existente en la herramienta sino a propiedades implícitas dentro del sistema software que se quieren mantener.

2.2.1. Funcionales

En cuanto a los requisitos funcionales, tendriamos principalmente los mencionados a continuación:

RF-1.– La herramienta, debido a que el objetivo de su uso es único por parte del programador a cargo de su desarrollo, solo tendrá un usuario registrado que debe acceder mediante un inicio de sesión para poder llevar a cabo el resto de funcionalidad.

RF-2.– La herramienta permite al usuario añadir un nuevo escaneo.

- **RF-2.1.–** El usuario podrá asignar un nombre de organización al escaneo.
- **RF-2.2.–** El usuario debe introducir el dominio en el que desea que se origine el escaneo.
- **RF-2.3.–** El usuario especifica además una configuración dentro de las que incluya la aplicación para utilizar sobre el escaneo.

RF-3.– La herramienta permite al usuario añadir diferentes tipos de configuración para los escaneos.

- **RF-3.1.–** El usuario podrá darle un nombre a la configuración creada.
- **RF-3.2.–** El usuario podrá especificar cuales de las tareas disponibles se quieren establecer en esa configuración.

RF-4.– Deben existir, al menos, las siguientes tareas a realizar sobre los escaneos:

- **RF-4.1.–** Enumeración de subdominios.
- **RF-4.2.–** Escaneo de puertos.
- **RF-4.3.–** Descubrimiento de hosts web.
- **RF-4.4.–** Descubrimiento de vulnerabilidades

RF-5.– El sistema permite al usuario elegir las herramientas a utilizar en la tarea de enumeración de subdominios.

RF-6.– La herramienta debe permitir al usuario editar un escaneo una vez creado.

RF-7.– La herramienta debe permitir al usuario lanzar múltiples escaneos de forma simultanea y paralela.

RF-8.– La herramienta permitirá al usuario consultar el estado actual de cualquiera de los escaneos, notificando los errores en caso de haber ocurrido.

RF-9.– La herramienta debe permitir al usuario programar diferentes escaneos en correspondientes intervalos de tiempo, para poder realizar una monitorización temporal y actualizada de cada organización.

RF-10.– La herramienta almacenará toda la información obtenida para cada uno de los escaneos en una base de datos local.

RF-11.– La herramienta mostrará los datos de los escaneos de forma tanto general como individual, y el usuario podrá consultarla y filtrarla de la manera que desee.

RF-12.– La herramienta permitirá al usuario crear nuevas reglas de detección de vulnerabilidades de forma sencilla y añadirlas de forma directa al funcionamiento básico de la herramienta correspondiente.

- **RF-12.1.–** El usuario podrá arrastrar un fichero para añadirlo directamente.
- **RF-12.2.–** El usuario podrá escribir a mano el código en formato yaml para ser añadido posteriormente.

2.2.2. No funcionales

Los requisitos no funcionales son igual o más importantes que los funcionales, ya que constituyen las propiedades básicas que debe cumplir la herramienta para tener un uso adecuado y accesible.

RNF-1.– La herramienta debe hacer un uso eficiente de los recursos que dispone para ejecutar todas las tareas en el menor tiempo posible.

RNF-2.– Siempre que se pueda, se deberá paralelizar el trabajo para aumentar el rendimiento de la herramienta al llevar a cabo múltiples tareas de forma simultánea.

RNF-3.– La herramienta debe ser extensible en todo momento de cara a poder implementar futuras mejoras después de la finalización del propio proyecto.

RNF-4.– La herramienta debe tener un soporte de base de datos para almacenar toda la información de interés de los escaneos del usuario de forma persistente.

RNF-5.– La herramienta debe mantener una ejecución persistente en el tiempo de cara a realizar la monitorización adecuada para las diferentes organizaciones.

RNF-6.– La herramienta tendrá un sistema local de almacenamiento de datos a través de una estructura de carpetas y ficheros, de forma que si en algún punto se elimina por error un escaneo y toda su información correspondiente, se podrá recuperar con una acción sencilla por parte del usuario.

RNF-7.– La herramienta debe estar optimizada para poder consultar el gran volumen de datos que se va a manejar en los distintos apartados de la aplicación en el menor tiempo posible.

RNF-8.– La herramienta debe tener un alto grado de usabilidad por parte del usuario, y todas las acciones que pueda realizar deben ser claras y directas.

RNF-9.– El tiempo de respuesta debe ser el mínimo posible, delegando la carga de datos a tareas en segundo plano en el caso de ser necesario.

RNF-10.– El sistema mantendrá los resultados de escaneos anteriores de forma acumulativa, añadiendo únicamente los datos nuevos.

2.3. Arquitectura

El último paso dentro de la etapa de diseño es definir la arquitectura que va a componer la herramienta, especificando de la mejor forma posible todos sus componentes y las tareas que llevan a cabo. Se buscará aprovechar al máximo los recursos disponibles, además de utilizar todas las herramientas que se han ido comentando en los pasos previos, para cumplir con las tareas y requisitos establecidos. En la figura 2.3 se puede apreciar la estructura con los 4 componentes principales que componen la arquitectura de la herramienta.

A continuación, vamos a detallar por partes cada uno de los componentes principales que aparecen en el esquema anterior, justificando su uso y la utilidad que tienen de cara a componer la arquitectura de la herramienta para este trabajo.

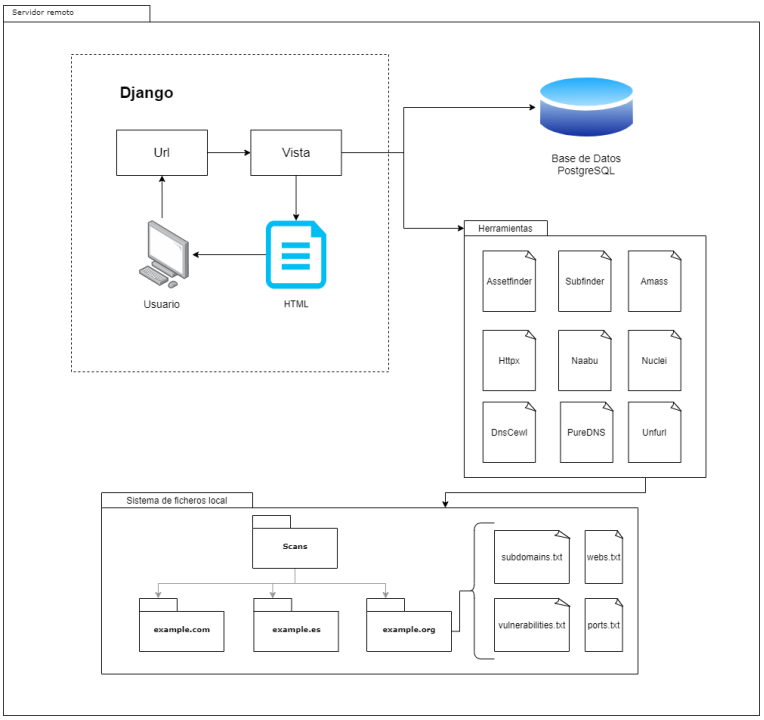


Figura 2.3: Diseño de arquitectura planteada para la herramienta

2.3.1. Django

La primera decisión de cara a la arquitectura fue el entorno y lenguaje en el que se quería llevar a cabo todo el desarrollo. Por un lado, habría que elegir un entorno o framework que permitiese agilizar el proceso de construcción de servicio web, con un diseño MVC para la solicitud de urls y vistas. Además, el objetivo era poder aprovechar al máximo conocimientos ya adquiridos a lo largo del grado estudiado. Por todo esto, la decisión principal fue utilizar Django.

Django es un framework de desarrollo web en Python que aplica el diseño modelo-vista-controlador, y que permite agilizar el proceso de construcción de un servicio web. La decisión se toma ya que se trata de un framework que se ha utilizado en trabajos del grado, y que como programador conozco en detalle. Además, se basa en Python, un lenguaje de alto nivel que permite programar a un ritmo más rápido que con otros lenguajes. Dentro de las posibilidades planteadas, también existía la posibilidad de utilizar Flask, un framework más sencillo que Django, pero que también lo he tratado a lo largo del grado y que se utiliza para la creación de aplicaciones Web. El hecho de haber elegido Django en vez de éste es sencillo de entender, ya que el motivo principal era el hecho de que al ser un framework más completo, simplifica muchas tareas que con Flask habría que hacer de forma manual, y esto ahorra mucho tiempo, ya que hay que recordar que el objetivo está más enfocado a la funcionalidad que se quiere conseguir que al propio desarrollo de un sistema web. También, Django tiene un buen soporte de base de datos, lo que nos permite también exprimir al máximo esa característica, la cual detallaremos posteriormente.

Por lo tanto, el servidor se programará en Python, realizando una sencilla tarea de, por un lado declarar los templates en código HTML que se cargarán tras las solicitud de una url, pasando antes por la vista correspondiente que realice la tarea lógica correspondiente, ya sea la carga o edición de datos, o la ejecución de alguna subrutina relacionada con las herramientas que se van a utilizar. Además, la ventaja que existe al utilizar este lenguaje de programación es el hecho de que será relativamente sencillo utilizar sistemas de paralelización de tareas con hilos, o incluso un sistema de programación temporal de tareas con alguna librería como Celery, en la cual entraremos más en detalle en el contexto adecuado de este documento.

2.3.2. Sistema de ficheros y herramientas

Ya se ha comentado numerosas veces que se utilizarán una serie de aplicaciones de código abierto que permiten realizar algunas de las tareas que componen los diferentes escaneos que defina el usuario. La mayoría de este tipo de herramientas en general —no únicamente las que se han comentado— suelen funcionar muy bien sobre sistemas operativos como Unix o Mac OS, y tienen normalmente modos de ejecución muy parecidos, además hay que tener en cuenta algo que se ha mencionado previamente al enumerar cada una de ellas, y es que hay varias que provienen de los mismos creadores, lo que implica que tienen métodos prácticamente idénticos a la hora de lanzarse. Sin embargo, lo realmente importante es que todas estas herramientas suelen escribir los resultados en algún tipo de salida estándar, o en su defecto, en un fichero especificado, lo cual servirá para tener un sistema adicional de persistencia de resultados aparte de la base de datos. Todo esto se va a organizar a nivel de scripting en bash.

La organización de ficheros va a ser sencilla, se tendrá una carpeta general con los resultados de todos los escaneos, y dentro una serie de carpetas cuyo nombre se establecerá a partir del dominio principal con el que se ha iniciado el escaneo. De esta forma, se tendrá dentro de cada carpeta de dominio los resultados obtenidos para cada una de las tareas, en formato de fichero. La estructura interna de ficheros que se ve en el diagrama está simplificada, pero se mantendrá un versionado de resultados para tener en todo momento el mayor número de ellos, es decir, la unión entre escaneos antiguos y nuevos, ya que habitualmente los activos de un sistema van variando con el tiempo. Las herramientas por lo tanto tendrán su modo de ejecución establecido en unos scripts, los cuales se ubicarán en una carpeta dentro del proyecto, y todos estarán programados para que el formato de destino sea correspondiente a este esquema mencionado. Una de las utilidades de Django en este aspecto también es la posibilidad de establecer las rutas a cada carpeta dentro de las variables de su fichero settings.py, lo cual permite generalizarlo de una forma muy cómoda.

2.3.3. Base de datos

La base de datos será el soporte principal para la persistencia de información dentro de la herramienta. Hoy en día es casi imprescindible tener este tipo de soporte incorporado a tu sistema web, y es por ello que no faltará dentro de este desarrollo. Al ser una herramienta cuyo único usuario será el propio desarrollador, ya que su objetivo es esencialmente didáctico, se mantendrá como base de datos local, lo cual quiere decir que no se utilizará un servidor únicamente para la gestión de base de datos. Django mantiene dentro de su metodología una fácil gestión con la base de datos mediante una vinculación dentro del fichero settings.py, y la declaración de modelos y atributos que se transcriben directamente a tablas y columnas. Además, la lectura y escritura de modelos dentro de las vistas es realmente intuitiva y directa, lo cual otorga un gran nivel de transparencia de acceso.

El siguiente paso es definir el esquema de base de datos que se va a implementar, y para ello se realiza un diagrama entidad-relación, el cual se puede ver en la figura 2.4. El elemento principal

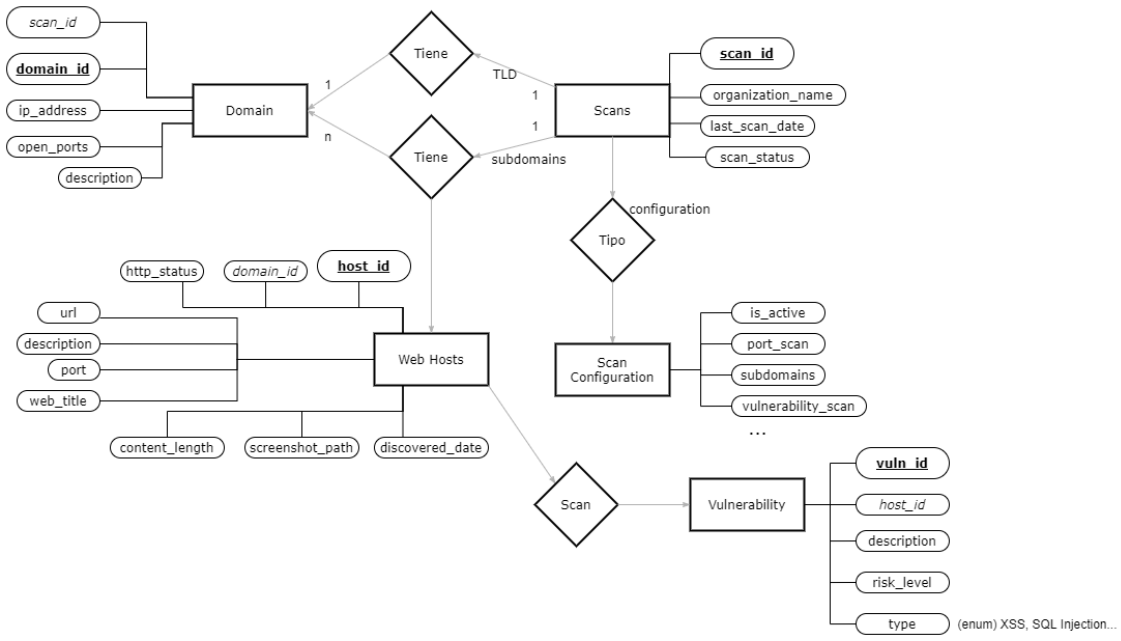


Figura 2.4: Diagrama Entidad-Relación para la base de datos de la herramienta

de este esquema es el escaneo, que contiene un **id**, un **nombre de organización** y un **estado**. Los escaneos tendrán una configuración asignada que podrá ir cambiando en el tiempo. Dichas configuraciones se mantendrán también en la base de datos para poder mantener todas las que el usuario haya ido creando para su utilización. Además, un escaneo tiene una relación directa con los dominios que contiene. Existe un dominio que será el original del escaneo, el cual se introduce al añadirlo, y el resto serán los que descubran las herramientas pertinentes. Los dominios a su vez tienen una relación con los hosts web que se vayan encontrando con la herramienta httpx, la cual obtiene una serie de parámetros de interés para los servicios web. Por último, tenemos las vulnerabilidades las cuales se detectarán en primera instancia a partir de los hosts web guardados para un escaneo. Todo esto

tendrá una evolución, ya que si en un futuro se incluye el análisis de vulnerabilidades sobre otro tipo de servicios, el esquema variará. Sin embargo conceptualmente para plantearlo de forma inicial antes del desarrollo, es lo suficientemente completo.

2.3.4. Servidor remoto

Todos los componentes que hemos mencionado anteriormente deben funcionar de forma fluida y sin interrupciones para asegurar la correcta monitorización de los sistemas. Todo esto conlleva una carga muy amplia de trabajo sobre la máquina en la que se encuentre, por lo que necesitamos garantizar la disponibilidad de la herramienta. Además, también hay que tener en cuenta que los sistemas de organizaciones, por mucho que permitan realizar con total libertad los escaneos sobre los dominios pertenecientes al scope, al final existen servicios de terceros como Akamai o Cloudflare que son CDNs que en ocasiones pueden llegar a banear IPs que estén realizando algún tipo de actividad inusual, como la que se suele realizar al lanzar escaneos de este tipo. Si el *baneo* de la IP fuese a una máquina personal, el usuario no podría disfrutar de forma natural de esos servicios —de hecho, adelantándonos ligeramente a los contenidos, durante una fase de pruebas con la máquina personal sobre algunos dominios de PlayStation, se detectó un *baneo* de forma que el usuario después no tenía permitido el acceso a algunos de sus servicios— por lo que también hay que intentar separar el ámbito de pruebas. Otro problema es el hecho de que los *routers* tienen un ancho de banda limitado, lo cual no es del todo óptimo si se quieren lanzar múltiples escaneos con diferentes herramientas de forma simultánea —algunas como nuclei que tiene una tasa muy alta de peticiones por segundo— ya que eso provocaría un cuello de botella y por lo tanto una denegación de servicio al propio router local.

Todos estos pequeños inconvenientes nos llevan a la decisión de realizar todo el desarrollo sobre un servidor remoto. Se necesita un servidor que esté alojado en una infraestructura con un alto porcentaje de SLA y de ancho de banda, que sea capaz de estar en funcionamiento las 24 horas del día. La contratación del servidor es en Contabo, un proveedor de VPS que oferta diferentes servidores con distintas características. Las propiedades del servidor contratado son:

- Cores: 8
- Memoria RAM: 30GB
- Almacenamiento: 800GB SSD
- Ancho de banda: 600 Mbits/segundo
- Sistema Operativo: Ubuntu 18.04

La decisión de Ubuntu 18.04 se debe a que la mayoría de herramientas que se van a incorporar son compatibles con éste sistema operativo, ya que están programadas en Golang o Python, por lo que con este tipo de entorno nos aseguramos un despliegue lo más ágil posible. Con esto finalizamos el capítulo de diseño y pasamos al apartado de implementación, donde se va a estructurar el trabajo desde un punto de vista mucho más especializado.

3

Implementación

La implementación de una herramienta de estas características se puede realizar dividiendo los objetivos técnicos que se necesitan. Por un lado, se puede comenzar estableciendo el framework principal que ya se ha comentado, Django. Luego, se puede incluir una implementación gráfica sencilla para darle rápidamente un aspecto atractivo que le otorgue cierto nivel de usabilidad, luego entrar a aspectos más técnicos como la inclusión del sistema de base de datos estableciendo su esqueleto principal y los módulos esenciales que contienen las distintas tareas que se querrán realizar. Por último, unificar todo de forma fluida de cara a obtener la funcionalidad finalmente deseada.

A continuación pasaremos a entrar en detalle en cada una de las etapas principales del desarrollo comentando el progreso y la evolución que ha ido siguiendo el código para obtener el resultado final.

3.1. Comienzos en Django

El primer movimiento en la implementación, como se ha mencionado anteriormente, se trata de iniciar el proyecto en Django, y establecer los ajustes generales para poder pasar al desarrollo de la aplicación. El primer paso es en ocasiones, el más complicado: escoger un nombre para la herramienta. Tratándose de una herramienta cuyo objetivo final es el descubrimiento de vulnerabilidades, en un ataque de originalidad se pensó el nombre **Vultec**, una especie de acrónimo de 'Vulnerability Detection', con el cual se hará referencia a la herramienta de ahora en adelante.

Vultec se inicia como cualquier otro proyecto de Django, con los comandos iniciales de creación de proyecto y aplicación. Dichos comandos realizan una generación automática de ficheros y carpetas que componen el esqueleto de todo el sistema, y se pueden ver a continuación. Cabe mencionar que para realizar tanto esta fase del desarrollo como el resto de objetivos relacionados con el framework se ha seguido la documentación en el libro *Tango with Django* [15] y la documentación web oficial [16].

```
django-admin.py startproject Vultec
```

```
python3 manage.py startapp scanner
```

El primer comando construye el esqueleto general del proyecto, creando una carpeta con el nombre Vultec, y que contiene los ficheros que se pueden ver en la figura 3.1. El fichero esencial es manage.py, el cual permite realizar numerosas acciones siendo la más relevante el lanzamiento de la aplicación mediante el comando *runserver*.

```
Vultec/
    manage.py
    Vultec/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

Figura 3.1: Generación del comando startproject de Django

Otro fichero de suma importancia es el fichero de settings.py, ya que contendrá la configuración global de todo el proyecto, y será el lugar donde se establecen parámetros importantes como algunas rutas de interés —en el caso de Vultec son la ruta a los templates, la ruta a la carpeta de contenido estático como imágenes, ficheros de estilo o JavaScript, la ruta a los scripts que contienen la lógica de lanzamiento de tareas, y por supuesto, la ruta a los resultados obtenidos tras cada escaneo—. También, contiene configuración de base de datos, la cual se detalla en el contexto adecuado de este documento, y otros detalles como las rutas de redirección. Estas rutas de redirección van de la mano con el fichero urls.py, otro pilar fundamental en la estructura de este framework, pues es el intermediario entre las solicitudes del cliente y el acceso al manejador en el servidor. El fichero urls.py se encarga de mapear las rutas solicitadas con los controladores pertinentes en el caso de estar registrados, para que luego se pase a realizar la tarea lógica correspondiente en el lado del servidor.

A continuación, dentro de la carpeta raíz de Vultec, procedemos a lanzar el segundo comando que hemos visto previamente, el cual nos sirve para crear aplicaciones dentro del proyecto. En general, los proyectos en Django se suelen realizar de cara a contener múltiples aplicaciones que funcionan ya sea de forma conjunta o independiente, sin embargo, por simplicidad de desarrollo ya que, como se incluye en los objetivos, no es el foco de atención dentro del proyecto, realizaremos una única aplicación llamada scanner. En la figura 3.2 se pueden ver los ficheros generados tras el comando. De aquí podemos destacar el fichero models.py, el cual será la columna vertebral en cuanto al soporte y mantenimiento de los datos, pues define las tablas y propiedades que se han establecido a través del diagrama Entidad-Relación. Entraremos más en detalle en la siguiente sección. El fichero views.py se encarga de establecer la lógica esencial del servidor para realizar las tareas que solicite el usuario,

```
scanner/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

Figura 3.2: Generación del comando startapp de Django

y devolver las vistas en forma de template HTML conteniendo los datos correspondientes. El fichero admin.py contiene la configuración principal de administrador, y el fichero apps.py se registran las aplicaciones que existen dentro del proyecto. Por último esta el fichero de tests. Éste es un fichero muy útil cuando la lógica es compleja y restrictiva, ya que permite añadir casos de prueba para comprobar que todo funciona como es debido. Sin embargo, durante el desarrollo de este proyecto no se tendrá muy en cuenta ya que la lógica se basará en la lectura y registro de datos, junto con el lanzamiento de escaneos, por lo que no vamos a darle tanta importancia a posibles restricciones en cuanto a accesos del usuario a datos o vistas, o ejemplos similares. Es importante mencionar que a lo largo del desarrollo habitual se suelen incluir al menos tres ficheros adicionales que son el urls.py, el forms.py y managers.py, pero se mencionarán en el momento adecuado justificando su inclusión.

Con esto terminamos los conceptos básicos sobre la iniciación en Django, y pasamos a la implementación de la base de datos dentro del framework, aprovechando su soporte.

3.2. Implementación de Base de Datos

A continuación, se va a describir la implementación de la base de datos y los pasos necesarios hasta poder hacer un uso adecuado de ella con Vultec. El primer paso es abrir el fichero models.py, el cual hemos mencionado previamente, y empezar definiendo las clases que conforman las tablas de la base de datos. En el código 3.1 podemos ver un ejemplo para el modelo de un dominio. Por ejemplo, podemos ver que tendrá una referencia en forma de clave foránea al escaneo que pertenece, pero también tiene atributos propios como un nombre, una dirección IP, una serie de puertos que se hayan descubierto, y una variable booleana, start_domain, la cual esta incluida para indicar si se trata del dominio original sobre el cual se ha creado el escaneo. Esto es meramente por motivos de comodidad a la hora de mostrarle los datos de un escaneo al usuario.

Código 3.1: Modelo de dominio, con los atributos que se quieren almacenar.

```
class Domain(models.Model):
    name = models.CharField(max_length=100, null=False)
    scan = models.ForeignKey(Scan, on_delete=models.CASCADE, db_index=True)
    ip_address = models.CharField(max_length=100, default="")
    open_ports = models.CharField(max_length=500, default="")
    start_domain = models.BooleanField(default=False)

    def __str__(self):
        return self.name
```

Esto, se traducirá de forma directa en una jerarquía de tablas y relaciones, que permitirán hacer uso de la base de datos de una forma muy intuitiva en el código. Una vez se tienen los modelos establecidos, se deben aplicar las migraciones correspondientes, de forma que Django entienda que ese es el esquema que debe almacenar. Las migraciones se almacenan en la carpeta *migrations* de la aplicación. Los comandos para conseguir esto son los siguientes:

```
python3 manage.py makemigrations scanner

python3 manage.py migrate
```

Por último, solo queda crear una base de datos, la cual será local del servidor y privada, y establecer la configuración necesaria para que Django realice la conexión acorde, introduciendo el código 3.2 en el fichero settings.py.

Código 3.2: Configuración de base de datos de la aplicación.

```
DATABASES = {}
if os.getenv('SQLITE', False):
    DATABASES['default'] = {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
else:
    import dj_database_url
    DATABASES['default'] =
        dj_database_url.config(default='postgres://user:passwd@localhost:5432/database_name')
```

Es común aprovechar dos distintos entornos de base de datos; uno para la ejecución de tests, que suele ser en SQLite por defecto con Django, y otro para el uso habitual de la herramienta, que en este caso se ha elegido PostgreSQL por ser un gestor muy conocido y utilizado a lo largo del grado.

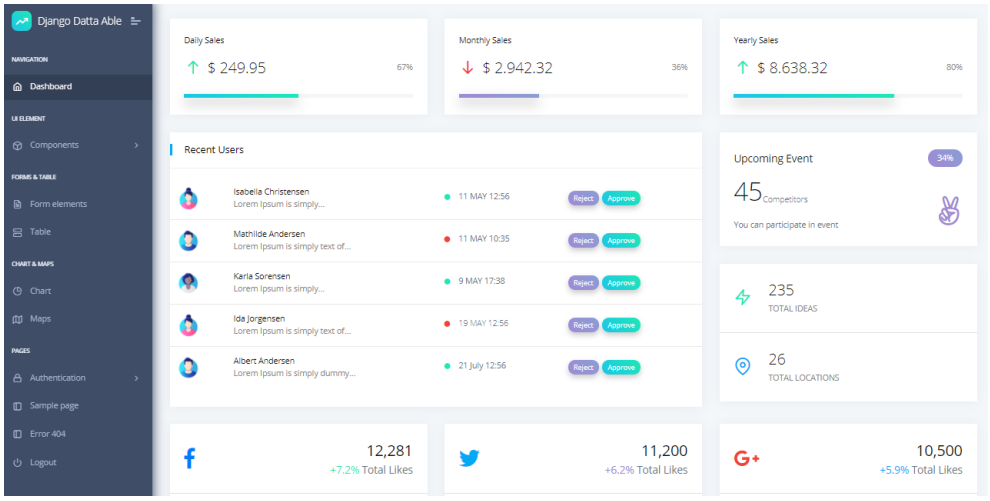
3.3. Desarrollo general

En este apartado, se va a detallar el desarrollo general de Vultec, y se dividirá en dos apartados: el apartado gráfico y el apartado de navegación. Por un lado, veremos el aspecto que ha adoptado la herramienta, y por el otro veremos las posibilidades que tiene el usuario a la hora de utilizarla y crear escaneos. A lo largo de esta sección se mostrarán algunas capturas de la herramienta en sus últimas versiones. Sin embargo se podrán ver imágenes de la interfaz completa en el Anexo C.

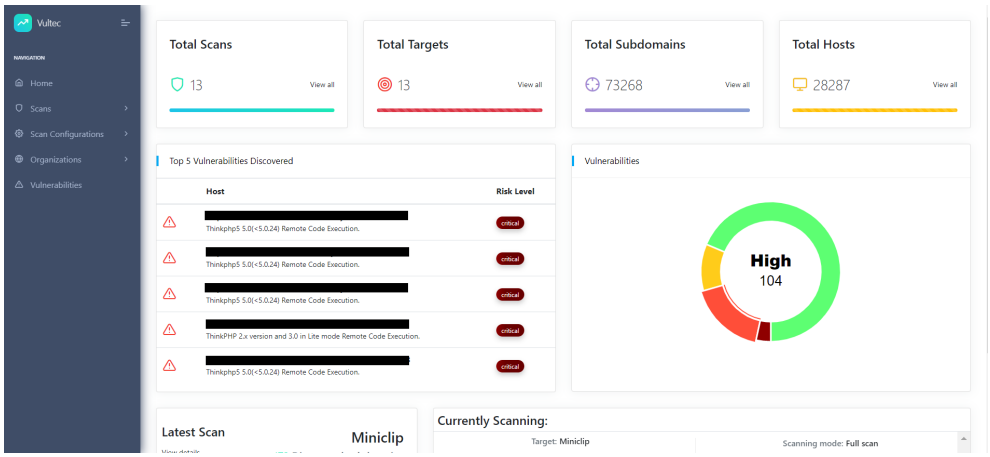
3.3.1. Implementación gráfica

La implementación gráfica no ha sido el foco de atención principal en ningún momento al plantear este trabajo, por lo que se busca gastar el mínimo tiempo para ello. Sin embargo sí es conveniente tener una interfaz sencilla y agradable, que muestre los datos de la forma adecuada y que sea cómoda de utilizar por parte del usuario. La solución más sencilla es buscar una plantilla de estilos en el formato *dashboard*, donde se tenga un menú lateral que nos de una navegación sencilla pero eficiente. Es por esto que se ha utilizado el generador automático de dashboards para Django, llamado Django Datta Able [17]. Se ha elegido esto ya que gracias a este generador de aplicaciones es trivial obtener una vista que contenga un menú lateral sobre el que se pueda navegar. De esta forma, se puede ver en la figura 3.3 por un lado el template original base que otorga la plantilla, 3.3(a), y como con los cambios necesarios se transforma en la interfaz que se desea 3.3(b). Esto nos permite aplicar todo el conocimiento de estilos que se ha adquirido a lo largo del grado o incluso mejorarlo, pero partiendo de una base lo suficientemente sólida como para que se ahorre el suficiente tiempo en este aspecto.

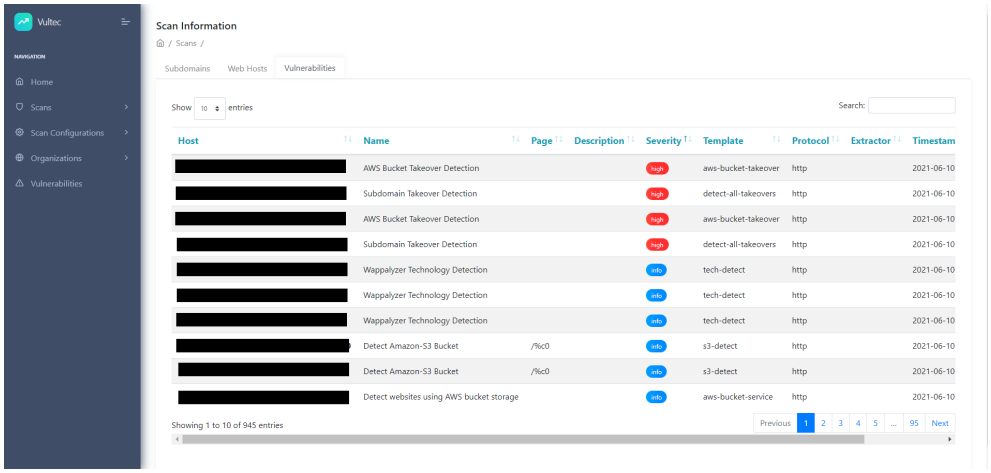
El desarrollo a partir de este punto en adelante es realmente intuitivo. Con la plantilla de estilos se hace mucho uso de Flexbox Grid, un sistema de CSS para ordenar los elementos de la interfaz por columnas, y como se puede apreciar se aplican bloques de 'cartas' para mostrar diferentes segmentos de información útiles para el usuario. Por otro lado, teniendo en cuenta que la volumetría de información que se obtendrá de cada escaneo es relativamente alta —y digo relativamente, ya que siempre dependerá de la cantidad de dominios encontrados— la mejor opción para representarla es en forma de tablas, por lo que se utiliza Bootstrap 4 y sus DataTables, como se puede ver en la figura 3.3(c), que muestra la tabla con la información de las vulnerabilidades encontradas para una organización a partir de su TLD. Éstas tablas permiten realizar de forma muy sencilla una paginación, ordenación por columnas o incluso filtro de búsqueda sobre la tabla, de cara a poder encontrar de la forma más rápida posible lo que realmente sea importante, ya que habrá que tener en cuenta que, en este caso de vulnerabilidades, hay algunas que son más graves que otras, por lo que evidentemente nos interesan más.



(a) Plantilla base de estilos descargada.



(b) Página de inicio de Vultec tras modificar estilos.

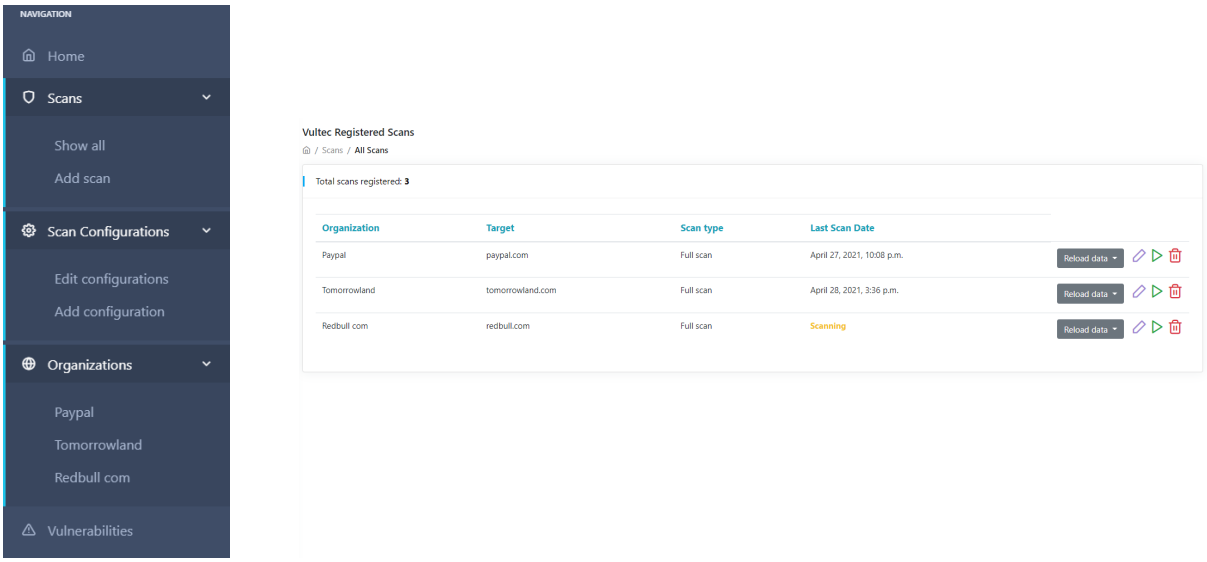


(c) Tabla de vulnerabilidades.

Figura 3.3: Interfaces de la aplicación.

3.3.2. Esquema de navegación

El esquema de navegación se ha mantenido lo más simplificado posible, de cara a tener las acciones de forma visible y rápida a través del menú lateral. Dicho menú se puede ver en la figura 3.4(a). Las acciones son muy simples, el usuario puede crear un nuevo escaneo o mostrar una tabla con todos ellos, donde se le muestran todas las acciones posibles que tiene sobre cada uno, como editarlo, eliminarlo o lanzarlo. Además, tiene la opción de añadir nuevas configuraciones, o editar las ya existentes. Dentro de las configuraciones habrá dos que existen por defecto y no serán modificables, que son la de *enumeración de subdominios* y el *escaneo completo*.



(a) Menú lateral de Vultec

(b) Tabla de escaneos de Vultec.

Figura 3.4: Navegación en Vultec.

Una de las ediciones que puede hacer el usuario sobre un escaneo es el hecho de modificar su configuración, eligiendo una de las existentes en la página correspondiente. Por último, existe el nivel de navegación en el que se pueden consultar todas las organizaciones que se han escaneado y ver los resultados correspondientes de cada uno, tanto en dominios descubiertos como webs y vulnerabilidades. Además, se ha considerado interesante incluir una única vista que muestre las vulnerabilidades de todo el sistema, para poder consultar todo lo que se ha ido encontrando con los diferentes escaneos, ya que al fin y al cabo ésto es lo importante que el usuario quiere aprovechar.

Este es el esquema de navegación principal, y como se ha mencionado anteriormente, en el Anexo C se podrán ver las vistas correspondientes a cada una de las partes que componen la interfaz de Vultec. A continuación se detallará el desarrollo realizado sobre los módulos de tareas y los escaneos de Vultec.

3.4. Módulos de tareas

En los módulos de las tareas se concentra la lógica que conecta Vultec con las herramientas de terceros. Si recordamos el esquema de arquitectura, ahora entramos a analizar la parte baja de éste. Existen en esta versión de la herramienta 4 módulos principales que veremos a continuación, y que esencialmente se enlazan mediante bash scripting y el código de Python en Django. A lo largo de los siguientes apartados se verán referencias simplificadas a algunas secciones de código, sin embargo el código completo se puede encontrar en el Anexo E. Además, también se pueden consultar los modos de instalación de las herramientas aplicadas en el Anexo D.

3.4.1. Enumeración de subdominios

El módulo de enumeración de subdominios es el más completo de los cuatro, ya que se encarga de utilizar múltiples herramientas para llevarse a cabo. Esto se debe a que es una de las etapas más importantes en el proceso de descubrir una vulnerabilidad, ya que cuantos más dominios se descubran más posibilidades habrá de encontrar una brecha.

Ya hemos mencionado previamente las herramientas que se van a utilizar en esta sección. La idea es que el usuario sea capaz de seleccionar la/s herramienta/s que quiera aplicar a su enumeración de subdominios a través de las posibles configuraciones, y que luego eso se traduzca para enviar las opciones al script. En la figura 3.5(a) podemos ver un ejemplo del fragmento de código con el que se comprueban las flags dentro del script, para asignar la herramienta correspondiente. De esta forma se leerán las herramientas elegidas y se pasa posteriormente a realizar la ejecución correspondiente como se puede ver en la figura 3.5(b). En dicha imagen, la variable '$PATH_RES' hace referencia a la ruta donde se almacenan los resultados de cada escaneo, y '$domain_selected' hace referencia al dominio origen del escaneo. Por último, se juntan todos los resultados obtenidos por cada una de las herramientas utilizadas en un fichero común, que luego será el que se lea desde el código en Python para guardar los resultados en la base de datos.

3.4.2. Escaneo de puertos

El escaneo de puertos consiste en una tarea muy sencilla, de hecho, se encarga de completar la información de puertos que se ha establecido para cada dominio en la base de datos de Vultec. Sobre los resultados obtenidos en la etapa de enumeración de subdominios, se lanza el comando que se puede ver a continuación, el cual lanza la herramienta Naabu sobre el fichero ya mencionado, y carga la salida en formato JSON al fichero *puertos.txt*. Muchas de las salidas que se quieren obtener serán en formato JSON ya que éste es fácilmente comprensible en el código en Python, por lo que se ha tomado dicha decisión.

```bash
for arg_n in $@
do
    case "$arg_n" in
        -A)
            if [[ "$tools" != *"Amass"* ]]; then
                tools+=" Amass"
            fi
            ;;
        -aF)
            if [[ "$tools" != *"AssetFinder"* ]]; then
                tools+=" AssetFinder";
            fi
            ;;
    esac
done
```

(a) Ejemplo de lectura de parámetros para la selección de herramientas en Bash

```bash
AssetFinder(){
    echo -e "${YELLOW}Running ${LIGHT_G}AssetFinder${YELLOW}...${NC}"
    assetfinder --subs-only $1 2>/dev/null > $PATH_RES/$domain_selected/subdomains_AssetFinder.txt;
}

SubFinder(){
    echo -e "${YELLOW}Running ${LIGHT_G}SubFinder${YELLOW}...${NC}"
    subfinder -d $1 -recursive -silent -t 200 -o $PATH_RES/$domain_selected/subdomains_SubFinder.txt &> /dev/null;
}
```

(b) Funciones correspondientes a las herramientas elegidas.

Figura 3.5: Secciones del script de enumeración de subdominios.

```
naabu -silent -iL subdominios.txt -json > puertos.txt
```

3.4.3. Descubrimiento de servicios web

El descubrimiento de servicios se ha llevado a cabo con la herramienta Httpx, como se ha mencionado anteriormente, y su uso es realmente sencillo. El comando se aplica, al igual que con la herramienta Naabu, sobre los resultados obtenidos en la fase de enumeración de subdominios. Con la herramienta podemos indicar los atributos que queremos obtener de cada host. En el caso del ejemplo que se puede ver a continuación, son los siguientes parámetros:

```
cat subdomains.txt | httpx -status-code -title -content-length -ip
-cname -web-server -tech-detect -ports 80,443 -json > webalives.txt
```

- status-code: El código HTTP de respuesta de la web.
- title: El título de la web.
- content-length: La longitud del contenido retornado.
- ip: La ip del host web.
- cname: El registro cname de la web.
- web-server: El tipo de servidor sobre el que está alojada la web.
- tech-detect: Las tecnologías sobre las que está construida la web

Además, está el parámetro *ports* para indicar los puertos que se quieren escanear y *json* para indicar el formato de salida. Adicionalmente se le pueden añadir parámetros para aplicar un timeout límite o indicar el número de hilos que se quieren utilizar.

3.4.4. Descubrimiento de vulnerabilidades

En cuanto al módulo de descubrimiento de vulnerabilidades, la herramienta principal es Nuclei. La herramienta se basa en un uso de reglas que están codificadas en lenguaje Yaml, que se obtienen de un repositorio llamado nuclei-templates, y se utilizan para lanzar una gran cantidad de peticiones a todos los servicios web que se quieran auditar. Acto seguido, las reglas precisamente comprueban si para la petición dada, se cumplen las condiciones necesarias como para que ese tipo de vulnerabilidad pueda existir dentro del sistema. La ejecución por lo tanto se puede ver a continuación.

```
nuclei -l webalives_scan.txt -t templates/cves -t templates/vuln...
-c 300 -bulk-size 500 -rate-limit 1200 -timeout 12 -retries 2 -json -o
nuclei_results.txt
```

Los parámetros que se utilizan son los siguientes:

- **t:** sirve para indicar una ruta de templates que se quiera aplicar. Se puede poner varias veces consecutivas.
- **c:** El número máximo de templates ejecutados de forma simultánea.
- **bulk-size:** El número máximo de hosts analizados de forma simultánea.
- **rate-limit:** El número máximo de peticiones por segundo.
- **timeout:** El timeout que se quiera establecer.
- **retries:** El número de veces que se reintentará una petición fallida.

La salida de nuevo será en formato json para optimizar lo máximo posible. Por supuesto, existen muchas más opciones, pero estas son las que se utilizan en la versión actual de la herramienta. Todas las opciones se pueden consultar en la documentación correspondiente de la herramienta. En el capítulo de pruebas, se entrará en detalle en el funcionamiento de los yaml, para entender la sintaxis e incluso analizar algunas de las reglas que hayan obtenido los mejores resultados.

3.5. Implementación de los escaneos

Tras la implementación individual de todos y cada uno de los módulos esenciales que componen la funcionalidad perteneciente al escaneo, llega el momento de conseguir que todo funcione de forma conjunta. Hay que tener en cuenta que las tareas de los escaneos en cierta forma dependen de los resultados de la tarea anterior, ya que es imposible obtener vulnerabilidades de servicios web si lo único que tenemos es la lista de subdominios, por ejemplo. Es por esto que en el código —ejemplo

3.3— se deben comprobar los ajustes de configuración en orden, para ir ejecutando cada uno de los scripts. Para ello se ha establecido una variable en el fichero settings.py que indica la ruta en la que éstos se encuentran.

Código 3.3: Fragmento de código en el que se ejecutan las diferentes tareas de forma secuencial.

```
1  if scan.configuration.subdomain_discovery:␣
2      if scan.configuration.tool_amass:␣flags␣+=␣"␣-A"␣
3      if scan.configuration.tool_assetfinder:␣flags␣+=␣"␣-aF"␣
4
5      subprocess.call([str(conf_settings.SCRIPTS_DIR)+'/find_subdomains.sh',␣'-d',␣target,␣flags])␣
6      chargeDomains(scan,␣target)␣
7
8  if scan.configuration.port_scan:␣
9      subprocess.call([str(conf_settings.SCRIPTS_DIR)+'/port_scan.sh',␣target])␣
10     chargePorts(target)␣
11
12 if scan.configuration.web_discovery:␣
13     subprocess.call([str(conf_settings.SCRIPTS_DIR)+'/web_alives.sh',␣target␣])␣
14     chargeWebs(scan,␣target)␣
15
16 if scan.configuration.vulnerability_scan:␣
17     subprocess.call([str(conf_settings.SCRIPTS_DIR)+'/vulnerabilities.sh',␣target␣])␣
18     chargeVulnerabilities(scan,␣target)␣
19
20 scan.scan_status␣=␣ScanStatus.FINISHED␣
```

Para realizar las llamadas a los scripts se ha utilizado la librería subprocess de Python, y tras cada una de las ejecuciones se tiene una funcion con prefijo *charge*<datos> que se encarga de leer los ficheros de texto y cargar los resultados en la base de datos. En el código 3.4 se puede ver el ejemplo sencillo para la carga de subdominios, los cuales se escriben cada uno en una linea en el fichero subdomains.txt. En esto consiste la realización de escaneos junto con las tareas que ya hemos visto, y a continuación entramos en un apartado en el que se van a ver algunas dificultades que se han ido encontrando a lo largo del proceso de desarrollo, y las soluciones aplicadas.

Código 3.4: Fragmento de código en el que cargan los subdominios del ficehro a la base de datos de Django.

```
1  def chargeDomains(scan,␣target):␣
2      with␣open(conf_settings.RESULTS_DIR+'/'+target+'/subdomains.txt',␣'r')␣as␣fin:␣
3          lines␣=␣fin.read().splitlines()␣
4          for␣line␣in␣lines:␣
5              Domain.objects.get_or_create(name=line,␣scan=scan)␣
```

3.6. Dificultades encontradas

El desarrollo en general de una herramienta de estas características no es especialmente sencillo, por lo general lleva numerosas horas montar toda la infraestructura y conseguir hacer una aplicación web sencilla a la vez que potente, pero hay ocasiones en los que se presentan problemas que realmente se pueden considerar como graves si de un desarrollo profesional se tratase, y se ha tenido que aplicar trabajo extra para descubrir el origen del problema y actuar de la forma adecuada.

3.6.1. Rendimiento

Un problema evidente era el problema de rendimiento a la hora de cargar las páginas de información de escaneos. Estas páginas consisten en 3 pestañas diferentes las cuales muestran todos los subdominios, webs y vulnerabilidades guardados para un mismo escaneo, respectivamente. La información se mostraba en formato de tabla paginada, utilizando DataTables de Bootstrap 4.

El problema residía en que toda la carga de información se hacía del lado del servidor de forma síncrona, esto es, hasta que no se tuviesen todos los dominios, webs y vulnerabilidades, no se cargaba la página. Con escaneos sencillos esto no era ningún problema, sin embargo, en la figura 3.6 se pueden ver algunas medidas de tiempo para 3 escaneos distintos, con su respectiva cantidad de subdominios.

```
Time finding scan: 0.0025517940521240234
Time filtering targets: 0.016637086868286133
Time filtering webs: 1.9311561584472656
Time filtering vulnerabilities: 1.0753211975097656
```

(a) Escaneo 1: 1.091 dominios.

```
Time finding scan: 0.0193946361541748O5
Time filtering targets: 2.225745677947998
Time filtering webs: 86.45648336410522
Time filtering vulnerabilities: 40.11034345626831
```

(b) Escaneo 2: 49,406 dominios.

Figura 3.6: Tiempos de carga de datos.

Como se puede comprobar, las páginas con menos cantidad de datos tardan un tiempo considerable, sin embargo una página con alta volumetría tardaría más de dos minutos en cargar, algo que es inaceptable. Es por esto que se decidió cambiar la estrategia en la carga de datos, pasando a utilizar consultas asíncronas. De forma muy sencilla se puede incorporar una petición de Ajax a la creación de la tabla, de forma que la página pueda cargar al instante, y simplemente se establezca un texto de carga mientras se traen los datos desde el servidor. Además, también podemos paralelizar la carga de las tres tablas de forma que así se ahorre mucho más tiempo. En el código 3.5 se puede ver un

ejemplo de una petición de Ajax para cargar los datos sobre una de las tablas. El rendimiento tras esta mejora aumentó considerablemente, logrando que la página de datos cargase al instante.

Código 3.5: Fragmento de código en el que se realiza una petición asíncrona de ajax para obtener los datos de la tabla de dominios.

```
(document).ready(function()␣{␣
    ('.domains').DataTable({␣
        "ajax":␣{␣
            "url":␣'{%␣url␣"load_scan_domains"␣%}',␣
            "data":␣{␣
                'scan_id':␣'{{␣scan_id␣}}'
            },␣
            "dataType":␣"json"␣
        }␣
    });␣
```

3.6.2. Paralelización de tareas

Otra de las dificultades encontradas durante el desarrollo fue el hecho de necesitar realizar una paralelización de escaneos, ya que obviamente no es útil la herramienta si no se pueden aprovechar las propiedades del servidor para optimizar el uso de recursos. La solución para dicha utilidad se puede ver en el código 3.6, donde se ha aprovechado para utilizar la librería threading de Python para abrir hilos de ejecución en paralelo, que se encargan de manejar los diferentes escaneos. Además, hay

Código 3.6: Paralelización de la tarea de ejecución de scripts mediante threading en Python

```
t␣=␣threading.Thread(name='runScripts',␣target=runScripts,␣args=(scan,␣target),␣daemon=True)␣
t.start()␣
```

herramientas que necesitan un modo de ejecución especial para poder paralelizarse, como es el caso de Amass, con la que hay que utilizar la opción *-nolocaldb*, de forma que no utilice su base de datos local para guardar datos, ya que esto no permite utilizarla de forma paralela.

3.6.3. Pérdida de datos tras fallos

Un problema que ocurre de manera muy frecuente cuando se hacen pruebas para los escaneos mientras se programa el código es el hecho de que en alguna ocasión las herramientas obtengan los resultados de la forma apropiada, sin embargo por un cambio realizado en la parte del código que ha provocado una excepción dichos datos no se hayan guardado de forma apropiada en la base de datos. En ese momento el desarrollador tendría que volver a ejecutar los escaneos solo para poder guardar

correctamente los valores en la base de datos, cosa que es totalmente ineficiente ya que los escaneos suelen tardar bastante tiempo, y sin embargo los resultados sí están escritos en los ficheros. Es por esto que se añaden las opciones que se pueden ver en la figura 3.7 a la tabla de escaneos, con las cuales el usuario puede llevar a cabo el proceso de lectura y carga de datos de forma independiente a los escaneos.

Figura 3.7: Opciones de recarga de datos en la tabla de escaneos

De esta forma, si en algún punto hay fallos en el código, o incluso el desarrollador está probando cosas de forma manual en el servidor para mejorar tareas y obtiene resultados prometedores, podrá insertarlos en la herramienta con esta opción. Es evidente que en términos de una aplicación que fuese oficial no tendría sentido tener algo así, ya que el objetivo es que, tal y como ocurre en la versión final de la herramienta, no haya errores inesperados, sin embargo es una sencilla inclusión que ha agilizado mucho el funcionamiento

4

Pruebas y resultados

Nos adentramos al momento decisivo del trabajo realizado y del documento que han estado leyendo, pues a continuación detallaremos el proceso de pruebas que se ha realizado, y analizaremos los resultados obtenidos.

Las pruebas realizadas se han hecho sobre programas públicos de Bug Bounty que se pueden encontrar en las páginas de Hackerone y Bugcrowd.

4.1. Resultados generales

En primer lugar, vamos a detallar el número de resultados generales que se han obtenido en las pruebas a través de métricas que reflejen el trabajo realizado, para posteriormente analizar, dentro de las vulnerabilidades, cuales han sido las más interesantes.

- **Número de escaneos realizados:** Se han realizado un total de **31** escaneos con Vultec sobre diferentes organizaciones obtenidas en los programas de Bug Bounty como PayPal, Netflix, Nintendo y Apple entre otras.
- **Número de subdominios detectados:** Se han detectado un total de **125.032** subdominios con Vultec en todos los escaneos realizados. Esto predispone una mayor posibilidad de detectar vulnerabilidades al haber encontrado una gran cantidad de dominios.
- **Número de servicios web descubiertos:** Se han descubierto un total de **55.873** servicios web con Vultec en todos los escaneos realizados, sobre los cuales se han lanzado las reglas de detección.
- **Número de vulnerabilidades encontradas:** Se han detectado un total de **45.211** vulnerabilidades con las distintas reglas de detección de Nuclei, tanto las originales como las personalizadas que se han incluido de forma manual, de las cuales:
    - **32** son de riesgo **crítico**.
    - **124** son de riesgo **alto**.
    - **73** son de riesgo **medio**.
    - **575** son de riesgo **bajo**.
    - El resto son de tipo informativo, lo cual no valora ningún tipo de riesgo sobre el activo.

4.2. Análisis de vulnerabilidades detectadas

A continuación vamos a analizar una parte de las muchas vulnerabilidades descubiertas, en concreto veremos las más interesantes a efectos técnicos, y se detallará la regla que ha descubierto cada una de ellas. Las evidencias para estas vulnerabilidades descubiertas se podrán encontrar en el Anexo F, así como sus reglas de detección.

Umbraco CMS LFI/RCE: Crítica

Umbraco CMS incluye un paquete *ClientDependency* que es vulnerable a una inclusión de archivos locales (*Local File Inclusion*) en la instalación por defecto a través del recurso DependencyHandler.axd.

Esta vulnerabilidad permite a un atacante acceder de manera externa a un fichero **web.config** (a priori accesible únicamente por un usuario con acceso al sistema). El fichero web.config se utiliza para almacenar información sensible sobre la configuración de aplicaciones .NET como por ejemplo:

- Credenciales para configuración de base de datos.
- Claves de autenticación (API Keys, tokens, etc.).
- Configuración sobre servicios (SMTP, Azure, AWS).
- Machine Key, configura los algoritmos y claves utilizados para el cifrado y descifrado. Esta información es especialmente sensible dado que su exposición permite, en algunos casos, ejecutar código remoto y tomar el control del sistema vulnerable.

Esta vulnerabilidad fue detectada en varias de las organizaciones, pudiendo extraer información sensible del servidor y ejecutando código de manera remota en los sistemas vulnerables.

Laravel .env File Accessible: Crítica

Existe una vulnerabilidad de fuga de información que afecta al framework Laravel hasta la versión 5.5.21, mediante la cual un atacante remoto puede obtener información sensible (como contraseñas de uso externo) a través de una petición directa al archivo /.env. Este fallo de configuración corresponde a una vulnerabilidad conocida de Laravel y asociada al CVE (https://www.cvedetails.com/cve/CVE-2017-16894/) que podría incluso permitir la ejecución de código de forma remota bajo ciertas circunstancias.

Este fallo fue detectado por la herramienta en un total de 2 activos, donde se exponía información sensible como por ejemplo:

- App Key utilizada por el framework Laravel.
- Credenciales de base de datos MySQL.
- Credenciales de servicio de correo (SMTP).
- Credenciales de AWS (access key y secret access key).

Generic SSRF Detected: Alta

La vulnerabilidad de *Server-Side Request Forgery* (conocida comúnmente como SSRF) es una vulnerabilidad web que permite a un atacante inducir a la aplicación (del lado del servidor) a realizar peticiones HTTP a un dominio arbitrario de su elección.

En los ejemplos típicos de SSRF, el atacante puede hacer que el servidor se conecte consigo mismo, o con otros servicios basados en la web dentro de la infraestructura de la organización, con aplicaciones internas en la nube o con sistemas externos de terceros.

Esta vulnerabilidad fue detectada en varias de las organizaciones. En ambos casos, la vulnerabilidad permitía interactuar con un servicio interno de Amazon Web Services (AWS Metadata Instance), que a priori únicamente es accesible desde la propia red local. Este servicio almacena las claves de acceso (**AccessKeyId** y **SecretAccessKey**) de AWS, lo cual permite tomar el control y gestionar una cuenta así como toda la información contenida en ella (código fuente, información sobre clientes y servicios, etc).

La vulnerabilidad de SSRF se encontraba en un parámetro vulnerable 'url' explotable a través de /?url=http://169.254.169.254/latest/meta-data.

Apache Solr <= 8.8.1 Arbitrary File Read: Alta

Apache Solr es una plataforma de búsqueda empresarial de código abierto del proyecto Apache Lucene. Está escrita en Java y se ejecuta como un servidor de búsqueda de texto completo independiente dentro de un contenedor de servlets como Apache Tomcat o Jetty.

En versiones anteriores a la 8.8.1, existe una vulnerabilidad de lectura de archivos mediante la cual un atacante puede acceder a ficheros internos del servidor y por tanto acceder a información sensible.

Esta vulnerabilidad fue detectada en una organización que contaba con varios entornos de Solr obsoletos desplegados en el puerto 8983. A través del recurso /solr/apifront/debug/dump?stream.url=file :///etc/passwd&param=ContentStream ha sido posible acceder al fichero /etc/passwd que contiene información de los usuarios del sistema.

WordPress accessible wp-config: Alta

Otra de las vulnerabilidades más relevantes que se detectaron, fue una fuga de información en una de las aplicaciones web de una organización, que permitía acceder y descargar el fichero de configuración de wordpress **wp-config.php**.

Este fichero contiene la información de acceso a la base de datos, así como las claves de seguridad que controlan el cifrado de la información en las cookies. El fichero se encontraba expuesto públicamente en el directorio raíz de la instalación de wordpress (a través de un fichero temporal **wp-**

config.php.save que no había sido eliminado) y por tanto podía ser accedido por cualquier atacante para posteriormente utilizar toda la información sensible almacenada.

Entre la información obtenida, encontramos las credenciales de acceso a una base de datos MySQL así como las claves de autenticación del aplicativo.

Subdomain Takeover: Alta

Sin lugar a dudas, una de las vulnerabilidades ampliamente extendidas a día de hoy se trata de la toma de control de subdominios. Un subdomain takeover se produce cuando un subdominio apunta a una cuenta de hosting compartido o a un servicio de terceros que es abandonado por su propietario, dejando el endpoint disponible para reclamarlo.

La herramienta Vultec, ha sido capaz de detectar numerosas vulnerabilidades de subdomain takeover, permitiendo tomar el control de diferentes servicios (aws ec2, azure, heroku, github, etc) de múltiples organizaciones.

Una vez tomado el control de los subdominios, el atacante puede controlar su contenido y alojar contenido malicioso, desplegar campañas de phishing avanzadas para robar credenciales de acceso o cookies de autenticación, etc.

SymfonyProfiler information leakage: Media

Symfony es un popular framework de aplicaciones web PHP. Una de las reglas desarrolladas, permite a la herramienta detectar instancias de Symfony desplegadas en modo desarrollo. En esta configuración, Symfony habilita un componente de depuración llamado web profiler. Las versiones de Symfony anteriores a la 2.7.13 sufren una vulnerabilidad de divulgación de información remota cuando el recurso app_dev.php se encuentra habilitado.

Uno de los activos vulnerables detectados, revelaba información muy sensible a través del recurso /app_dev.php/_configurator/final de Symfony. Concretamente, se pudo obtener credenciales de la base de datos que alojaba información personal sobre clientes y pedidos, así como las claves de acceso a un servicio de correo SMTP que permitía conectarse y enviar correos electrónicos de manera ilegítima.

Git Config Disclosure: Media

Otra vulnerabilidad muy común, es la exposición pública de repositorios de Github, generalmente a través del recurso **.git/config**. Este endpoint permite a un atacante recuperar información sobre el código fuente, el historial de git y los commits, lo que podría revelar información sensible dependiendo de la información almacenada y los commits realizados.

Un atacante también podría extraer información sensible solicitando el directorio de metadatos

oculto que crea la herramienta de control de versiones Git. Los directorios de metadatos se utilizan con fines de desarrollo para hacer un seguimiento de los cambios de desarrollo en un conjunto de código fuente antes de que se envíe a un repositorio central (y viceversa). Cuando el código se traslada a un servidor en vivo desde un repositorio, se supone que se hace como una exportación y no como una copia de trabajo local, y de ahí este problema.

Este fallo ha sido detectado en varias organizaciones, lo que han permitido la obtención de información sensible como por ejemplo:

- Código fuente del aplicativo.
- Claves de autenticación, bases de datos, etc.
- Credenciales de servicios como Azure o AWS.

Esto resulta especialmente útil, ya que al tener acceso al código fuente, un atacante podría detectar fallos en la programación que permitieran la explotación de vulnerabilidades con un impacto más elevado como podrían ser: Inyecciones de código SQL o ejecuciones de código.

Apache Tomcat Open Redirect: Media

Las vulnerabilidades de .ºpen Redirect"surgen cuando una aplicación incorpora datos controlables por el usuario en el objetivo de una redirección de forma insegura. Un atacante puede construir una URL dentro de la aplicación que cause una redirección a un dominio externo arbitrario. Este comportamiento puede ser aprovechado para facilitar ataques de phishing contra los usuarios de la aplicación.

El peligro de esta vulnerabilidad radica en que el usuario que puede ser engañado, en el momento de acceder al enlace, verá que corresponde a una página de confianza, con una conexión segura, lo que hace más real este ataque. Esta vulnerabilidad supone un riesgo tanto para los usuarios como para la reputación de la empresa afectada.

Una de las empresas monitorizadas, la cual ofrece servicios a clientes internacionalmente, se encontraba afectada por una vulnerabilidad de .ºpen Redirect"debido a un fallo de configuración del servidor (Apache Tomcat). Por ejemplo, cualquier usuario que accediese al recurso /.dominio-malicioso.com, se vería automáticamente redirigido al dominio externo dominio-malicioso.com,

MySQL Dump Files: Media

Otro de los fallos más comúnmente detectados, se trata de la exposición de ficheros de respaldo o backup en el directorio raíz de las aplicaciones web.

Es habitual, que los desarrolladores realicen copias de seguridad de sus aplicaciones y posteriormente olviden eliminar o limitar la visibilidad de estos archivos, permitiendo a cualquier atacante acceder a ellos a través de la aplicación web sin restricción alguna.

Estos ficheros pueden encontrarse en directorios como por ejemplo: /db.zip, /backup.rar, /dump.sql, /source.zip, etc. La herramienta ha sido capaz de detectar varios ficheros de backup expuestos a Internet que incluían información sensible como por ejemplo:

- Código fuente del aplicativo. Lo cual permite identificar fallos en la programación así como acceder a credenciales almacenadas en dichos ficheros.
- Copias de seguridad de bases de datos.
- Ficheros de configuración con credenciales en texto claro.

4.3. Vulnerabilidades reportadas

En el apartado anterior, se han detallado algunas de las vulnerabilidades más importantes detectadas en las diferentes organizaciones a lo largo de estos últimos meses. Estas vulnerabilidades, fueron notificadas responsablemente en cada uno de los programas de bug bounty de las organizaciones afectadas.

En algunos casos, las vulnerabilidades fueron aceptadas y recompensadas económicamente. Otras sin embargo, fueron categorizadas como duplicadas, dado que otro investigador había detectado y notificado previamente dicha vulnerabilidad. En este apartado, analizaremos los resultados obtenidos tras la notificación de las vulnerabilidades descubierta así como las recompensas recibidas en cada caso.

Umbraco LFI/RCE - Esta vulnerabilidad fue reportada a un total de cuatro organizaciones. Tres de estos reportes fueron aceptados y uno de ellos marcado como duplicado. Tan solo una de las vulnerabilidades permitía la ejecución de código remoto, sin embargo, esta vulnerabilidad permitió acceder a información muy sensible de todas y cada una de las empresas afectadas, por lo que todos los reportes se clasificaron como riesgo crítico o alto.
La recompensa por este tipo de vulnerabilidades oscila entre los 1000€-3000€ por reporte.

Generic SSRF - Esta vulnerabilidad fue reportada a un total de dos organizaciones. Ambos reportes fueron aceptados por las organizaciones y por suerte, ninguno resultó duplicado. En una de las vulnerabilidades se logró acceder a la red interna de la compañía y comprometer la instancia de AWS sobre la cual estaba desplegada dicha aplicación y acceder al código fuente de todas las aplicaciones desarrolladas por la compañía.
La recompensa por este tipo de vulnerabilidades oscila entre los 1000€-5000€ por reporte.

Symfony Profiler Information Leakage - Esta vulnerabilidad fue reportada y aceptada en tan solo una de las organizaciones involucradas en este análisis. Dicho fallo permitía obtener las credenciales de un servidor SMTP por lo que el reporte finalmente se consideró

crítico. Adicionalmente, en este mismo activo vulnerable, se descubrió un recurso expuesto públicamente (/docker-compose.yml) que contenía unas credenciales de la base de datos interna. Se notificó a la empresa afectada pero dicho reporte fue finalmente descartado como duplicado.
La recompensa por este tipo de vulnerabilidades oscila entre los 500€-3000€ por reporte.

Subdomain Takeover - Como ya mencionamos anteriormente, esta es una de las vulnerabilidades que más detecciones ha tenido por parte de la herramienta. Esto se debe a que los activos de las empresas expiran y cambian constantemente, por lo que los subdominios asociados pueden llegar a ser controlados por los atacantes si estos no se configuran debidamente. Tras el análisis, se notificaron un total de cuatro vulnerabilidades de subdomain takeover, todas ellas aceptadas por las organizaciones afectadas.
La recompensa por este tipo de vulnerabilidades oscila entre los 250€-1500€ dependiendo del subdominio y/o servicio afectado.

Estas vulnerabilidades son tan solo un ejemplo de algunas de las vulnerabilidades notificadas a las organizaciones tras las detecciones obtenidas por parte de la herramienta. A día de hoy, algunos de estos reportes se encuentran todavía en investigación y en fase de mitigación por parte de las empresas afectadas.

La herramienta a día de hoy, sigue detectando y notificando fallos diariamente debido a la monitorización activa de subdominios, aplicativos web y vulnerabilidades así como la elaboración de nuevas reglas que permite detectar nuevos fallos de seguridad.

5

Conclusiones

En este último apartado concluiremos el trabajo desde un punto de vista personal y técnico, y además se expondrán las futuras mejoras que tendrá la herramienta tras la finalización de este proyecto.

5.1. Conclusiones

Tras varios meses de trabajo, se ha conseguido comprobar el hecho de que las organizaciones que poseen sistemas informáticos de cualquier extensión, necesitan de forma constante una monitorización y análisis de sus servicios. Esto es de vital importancia, y los procedimientos tradicionales como los ejercicios de *pentesting* o *red team* son las mejores opciones.

Sin embargo, ha quedado demostrado que es relativamente sencillo realizar un sistema que monitorice y analice de forma automatizada. Lo que realmente supone un reto, incluso para los más expertos, es el hecho de conocer las reglas que realmente puedan obtener los resultados deseados.

En cuanto al proyecto realizado, se ha obtenido una herramienta que cumple los objetivos establecidos: ayudar a alguien a iniciarse en el ámbito de la ciberseguridad, pero también tiene potencial como para convertirse en un buen soporte para agilizar el trabajo a un *pentester* experimentado, permitiendo que éste pueda personalizar las reglas de búsqueda y así mejorar sus descubrimientos.

Personalmente, puedo afirmar que se han cumplido con creces los objetivos establecidos al inicio del proyecto, algo que queda demostrado tras obtener algunos de los resultados tan prometedores como los que se han visto en el capítulo 4: *Pruebas y resultados*. Además, este trabajo ha abierto una puerta dentro de mis límites sobre la ingeniería informática, pues me ha iniciado en un área que es hoy en día, predominante.

Por último, a nivel personal he quedado satisfecho pues he sido capaz de realizar un trabajo constante, llevando a cabo un proceso de aprendizaje nuevo y aplicando los conocimientos que he ido adquiriendo a lo largo del grado a partir de diversas asignaturas o etapas de éste.

5.2. Trabajo futuro

Por supuesto, Vultec solo acaba de conocer su primera versión, pues algo en lo que se ha hecho especial hincapié a lo largo de todo este documento es el hecho de que la herramienta ha sido programada de cara a ser completamente extensible, y es evidente que se tenían ideas adicionales que finalmente no han llegado a ser implementadas para la finalización de este trabajo.

Sin embargo, a continuación se detallarán algunas de esas mejoras que recibirá Vultec en el futuro y se explicará la utilidad de cada una de ellas.

Programación de escaneos

Una mejora inmediata será el hecho de poder aplicar un sistema de monitorización real a la propia herramienta, asignando intervalos horarios o fechas para que los escaneos y análisis se realicen de forma automática y recurrente, sin necesidad de que el usuario lo haga de manera manual.

Esto es interesante ya que muchas veces, el primero que encuentra un fallo es el que se lleva la recompensa, y los activos dentro de una organización cambian con frecuencia en el tiempo.

Editor interno para reglas de detección

Sería conveniente que el usuario pudiese codificar las reglas desde un editor en la propia interfaz de la herramienta, para luego poder guardarlas a través de una orden en el sistema de carpetas de nuclei.

Generación automática de reportes

Al igual que el caso anterior, sería útil tener la opción de generar un reporte automático a partir de una vulnerabilidad encontrada. Muchas veces, cuando se encuentra una vulnerabilidad es necesario explotarla antes para comprobar si realmente existe exposición a información sensible, pero en ocasiones la propia regla demuestra que la vulnerabilidad existe, por ejemplo si fuese una detección de *Cross Site Scripting*, para cuyos casos sería eficiente tener esta opción a mano.

Uso de Axiom para lanzamiento de escaneos

Axiom es una herramienta que permite realizar un balanceo de carga en diferentes máquinas remotas de forma dinámica. Esto nos permitiría aumentar aun más el rendimiento y eficiencia al dividir por ejemplo, en 10 maquinas el trabajo de una tarea por partes.

Esta mejora aumentaría la disponibilidad, la eficiencia y el grado de paralelización que se podría adquirir con la herramienta. Además, el uso de Axiom conlleva un coste económico muy bajo (pagas

por tiempo de uso, una cantidad relativamente baja).

Uso de proxies con Nuclei

Otra mejora que se puede añadir es el hecho de utilizar proxies cada vez que se lanzan escaneos con Nuclei. Nuclei es una herramienta que genera una gran cantidad de tráfico, algo que puede ser detectado y baneado por alguna organización.

Al usar proxies, tendríamos siempre el tráfico encapsulado por una IP distinta cada vez, lo que permitiría asegurarnos que siempre obtenemos los mejores resultados posibles.

Inclusión de herramientas de *fuzzeo*

Las herramientas de fuzzeo como ffuf nos permiten realizar escaneos de directorios automatizados dentro de las organizaciones. Esto nos permite realizar una especie de mapeo de la jerarquía de recursos interna del servicio. Por ejemplo, podríamos tratar de detectar de manera automática paneles de administración (/admin, /administrador) o detectar ficheros de configuración (/config.php, /config.json) para su posterior análisis manual.

Inclusión de herramientas para obtener screenshots

Herramientas como webscreenshot nos permitirían dar información adicional sobre las aplicaciones web descubiertas, obteniendo y guardando en la herramienta una captura de pantalla de la respuesta obtenida. Esto en ocasiones puede dar información visual que no se obtiene de forma técnica en las respuestas de httpx, y que puede ayudar al pentester a encontrar una posible vulnerabilidad de forma manual o decidir, con un simple vistazo, que aplicaciones pueden resultar más interesante analizar manualmente.

Inclusión de herramientas de monitorización de repositorios públicos

Es muy habitual, la exposición de credenciales o información sensible a través de repositorios públicos en Internet. Una de las fuentes más habituales suele ser Github, utilizado principalmente para el desarrollo de aplicaciones de código abierto. Herramientas como gitGraber son muy útiles de cara a monitorizar fugas de información dentro de las organizaciones. Información expuesta como credenciales, tokens o api keys son muy valiosas de cara a realizar un trabajo manual de descubrimiento de vulnerabilidades, y sería interesante útil incluir dicho tipo de herramientas en Vultec.

Bibliografía

[1] P. Hudak., “Asset discovery: Doing reconnaissance the hard way,” 2018. (Ver).

[2] David, “Just another recon guide for pentesters and bug bounty hunters,” 2020. (Ver).

[3] S. Cyclewala., “The complete subdomain enumeration guide,” 2020. (Ver).

[4] Tomnomnom., “Assetfinder,” 2020. (Ver).

[5] ProjectDiscovery., “Subfinder,” 2021. (Ver).

[6] OWASP., “Amass,” 2021. (Ver).

[7] P. Hudak., “Subdomain enumeration: 2019 workflow,” 2019. (Ver).

[8] Codingo., “Dnscewl,” 2021. (Ver).

[9] D3mondev., “Puredns,” 2021. (Ver).

[10] Blechschmidt., “Massdns,” 2021. (Ver).

[11] Tomnomnom., “Anew,” 2019. (Ver).

[12] Tomnomnom., “Unfurl,” 2018. (Ver).

[13] ProjectDiscovery., “Httpx,” 2021. (Ver).

[14] ProjectDiscovery., “Naabu,” 2021. (Ver).

[15] “Tango with django,” (Ver).

[16] “Django documentation,” (Ver).

[17] A. Generator., “Django datta able,” 2020. (Ver).

[18] Bonino., “Lemonbooster,” 2020. (Ver).

[19] Yogeshojha., “Rengine,” 2021. (Ver).

[20] “Download and install go,” (Ver).

Apéndices

A

Glosario de términos

TLD: *Top Level Domain* o dominio de nivel superior, supone el nivel más alto en la jerarquía de dominios dentro de una organización.

Framework: En el contexto de desarrollo web, entorno de trabajo que sirve para dar soporte a un programador para la construcción de un programa, facilitando de antemano tareas estructurales.

MVC: Modelo-Vista-Controlador, patrón utilizado comúnmente en las arquitecturas software para separar diferentes partes dentro de un sistema. El modelo contiene los datos, el controlador la lógica, y la vista la interfaz.

Template: En el entorno de Django, consiste en el código HTML que se utiliza para construir las vistas de la aplicación.

URL: Parte de la identificación de los recursos web. Es el localizador del recurso, usado por los navegadores para acceder a un recurso único de un servidor.

Migración: En el contexto de bases de datos, la migración consiste en la transferencia de una cantidad de datos de un origen a un destino.

Dashboard: Modo de representación de datos en una página web para realizar un seguimiento ordenado de la información que se está utilizando.

Script/Scripting: En el contexto informático, consiste en una secuencia de comandos escritos para un formato de terminal específica (en este caso Bash) que se ejecutan de forma secuencial para realizar una tarea.

Bash: Interfaz de usuario de línea de comandos de UNIX.

Wordlist: Diccionarios de palabras que se suelen utilizar junto con herramientas de permutaciones para fuerza bruta, o para pruebas de detección con diferentes parámetros.

Wildcard: Patrones comunes en las URLs utilizadas a modo de expresión regular. Por ejemplo, es muy común ver la siguiente estructura: *.example.com, que aplica a todos los dominios por debajo de 'example.com'.

Transparencia de acceso: Refiere al grado de abstracción que tiene el programador a la hora de acceder a los distintos componentes de un sistema informático. Cuanto mayor es

la transparencia, más accesibles son los componentes, simulando en parte una unidad.

Disponibilidad: Grado de capacidad que tiene un sistema o servidor informático de mantenerse funcionando en cualquier momento. Cuando se ataca la disponibilidad de un servidor, por ejemplo con un ataque de Denegación de Servicio (DOS), éste no es capaz de atender a todos lo usuarios que requieren de su uso.

ISP: Proveedor de servicios de Internet. Empresa que brinda conexión a Internet a un usuario o cliente.

SLA: Acuerdo de nivel de servicio, escrito entre el proveedor y el cliente para acordar la calidad de dicho servicio.

Ancho de banda: Tasa de datos que se pueden transferir por segundo entre dos extremos de una red de comunicaciones.

Cuello de botella: Problema en las redes de comunicaciones que se produce cuando no existe el suficiente ancho de banda para transmitir la cantidad de datos que solicita enviar uno de los dos extremos de una comunicación.

Endpoint: Consiste en el extremo de una red de comunicaciones. Habitualmente suele referirse a servidores, portatiles o móviles.

Scope: Alcance establecido por una organización a la hora de analizar sus sistemas informáticos, para delimitar los activos que la propia organización desea exponer.

CDN: *Content Delivery Network* o red de distribución de contenido. Se trata de un conjunto de servidores cuyo objetivo es balancear la carga dentro de una red de comunicaciones, eliminando posibles cuellos de botella

Ban: En el contexto informático, se refiere a una restricción de cualquier grado al uso de algún tipo de servicio.

VPS: *Virtual Private Server* o servidor virtual privado, recomendable para proyectos en los que se espera generar una alta cantidad de tráfico, pues suelen tener un alto grado de disponibilidad y ancho de banda.

Python: Lenguaje de programación interpretado, multiplataforma que hoy en día se utiliza en una gran parte de las ramas de la ingeniería informática, incluyendo el desarrollo web.

JavaScript: Lenguaje de programación interpretado orientado a objetos, utilizado principalmente del lado del cliente en el desarrollo web.

HTML: *HyperText Markup Language*, un lenguaje de marcado de hipertexto que se utiliza para estructurar una página web y sus contenidos.

CSS: *Cascading Style Sheets* es un lenguaje de estilos utilizado para la presentación de documentos estructurados escritos en lenguajes de marcado como por ejemplo, HTML.

JSON: Formato de texto para el formato, representación e intercambio de datos.

Ajax: *Asynchronous JavaScript and XML* es una técnica de desarrollo web que se utiliza para generar peticiones asíncronas en desarrollo web desde el lado del cliente, utilizando JavaScript y XML.

Yaml: Formato de serialización de datos inspirado en lenguajes como Python, C o XML.

B

Herramientas de referencia

Como ya se ha mencionado anteriormente, existen múltiples herramientas que realizan una tarea similar a la que se ha implementado en este trabajo de fin de grado, sin embargo el objetivo era desarrollar una desde cero para elaborar un proceso de aprendizaje adecuado. En este anexo se revisarán dos de las herramientas que se han tomado como referencia, y se especificará que diferencias principales que tienen con la que se ha desarrollado en este proyecto.

B.1. LemonBooster

LemonBooster [18] es una herramienta de código abierto que, cito textualmente, *"...automatiza y supervisa gran parte de la fase de reconocimiento. Se ejecuta en un entorno web, y facilita la ejecución de herramientas"*. Su tarea principal será llevar a cabo la fase de enumeración de subdominios y activos de forma automatizada y monitorizada. La herramienta permite utilizar numerosas herramientas para lanzar escaneos y realizar una enumeración de subdominos, almacenando todos los resultados obtenidos en una base de datos de MongoDB. La herramienta podrá comparar resultados antiguos con los nuevos tras cada escaneo.

B.2. Rengine

Rengine [19] es un framework de reconocimiento de código abierto también, que permite configurar escaneos con distintas opciones para analizar dominios en busca de información. Utiliza múltiples herramientas para realizar las tareas de descubrimiento de subdominios, búsqueda de *endpoints*, y se encuentra en una versión preliminar de escaneo de vulnerabilidades. Utiliza herramientas de código abierto que permiten llevar a cabo cada una de las etapas diferentes dentro del proceso de reconocimiento, las cuales se definirán posteriormente en este propio documento.

B.3. Diferencias principales

Como se puede comprobar, estas herramientas cumplen una función muy similar a la que se plantea en este trabajo de fin de grado. Sin embargo, a continuación se especifican algunos puntos donde se muestran las influencias que se toman, así como otros en los que se buscarán objetivos distintos de cara a personalizar la herramienta. Sin embargo, cabe recalcar que un objetivo es el hecho de realizar este desarrollo de cara al aprendizaje, no a la innovación.

En cuanto a las influencias, podemos tomar los siguientes aspectos:

- La herramienta automatizará escaneos para obtener información.
- Utilizará múltiples herramientas de código abierto para llevar a cabo las distintas tareas.
- Se podrá configurar el tipo de escaneo según las tareas que se quieran incluir.

Las diferencias principales entre la herramienta que se ha planteado en este trabajo de fin de grado y las dos mencionadas previamente son las siguientes:

- Se incluirán técnicas de fuerza bruta y permutaciones para el descubrimiento de subdominios.
- Se podrán personalizar reglas para la detección de vulnerabilidades, para no utilizar únicamente las que vienen por defecto en las herramientas de terceros.
- Se incluirá el escaneo de puertos para reconocimiento de vulnerabilidades en servicios no web (versiones mas avanzadas)
- Se podrán programar los escaneos con intervalos de tiempo establecidos para mantener una monitorización constante
- Se utilizarán sistemas para distribuir la carga de trabajo de forma eficiente de cara a optimizar lo máximo posible el rendimiento en cada escaneo.

C

Imágenes de la interfaz

En éste apéndice se van a mostrar capturas de la interfaz generada para **Vultec**, no pudiendo mostrar fragmentos pues es información privada sobre organizaciones que no es legal mostrar de forma pública.

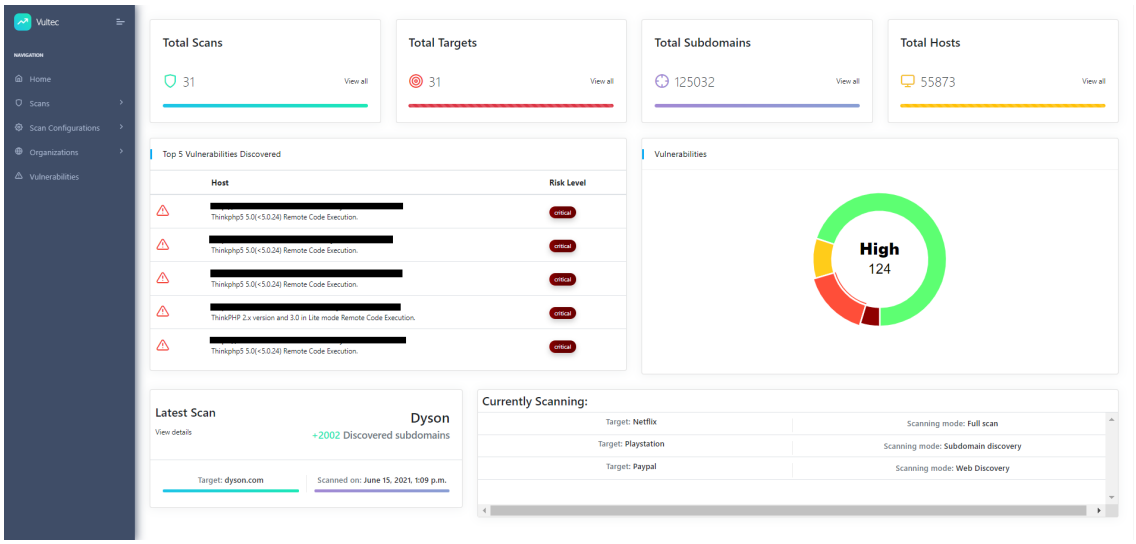


Figura C.1: Página principal de Vultec.

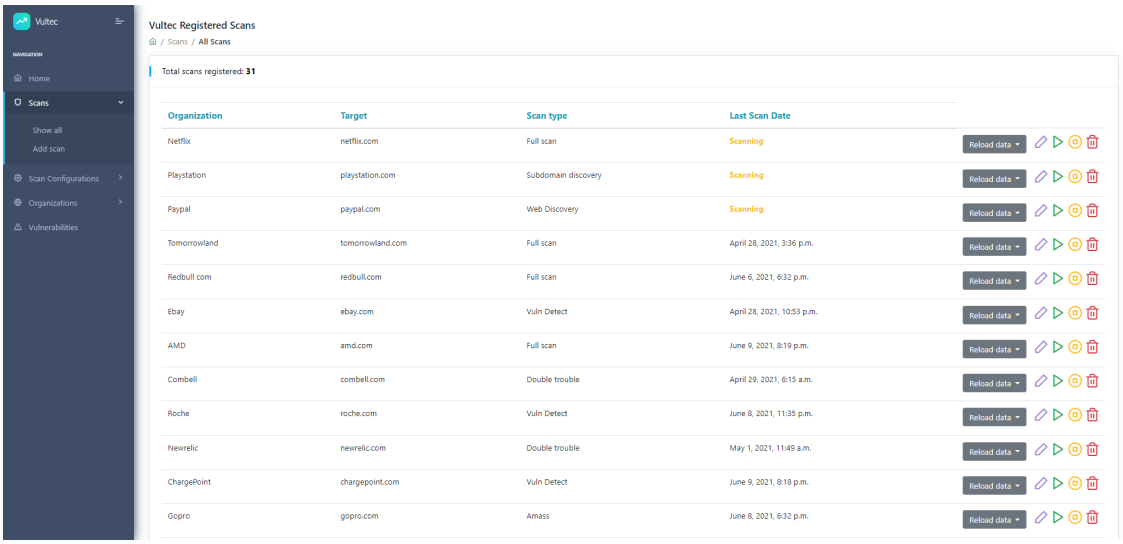


Figura C.2: Página que muestra todos los escaneos registrados en Vultec.

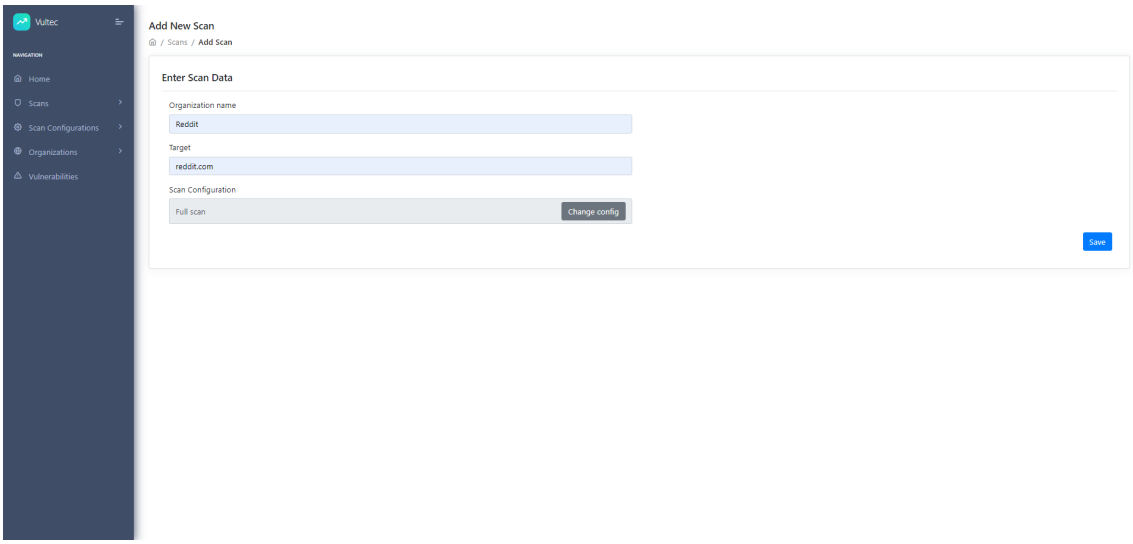


Figura C.3: Página para añadir un nuevo escaneo a Vultec.

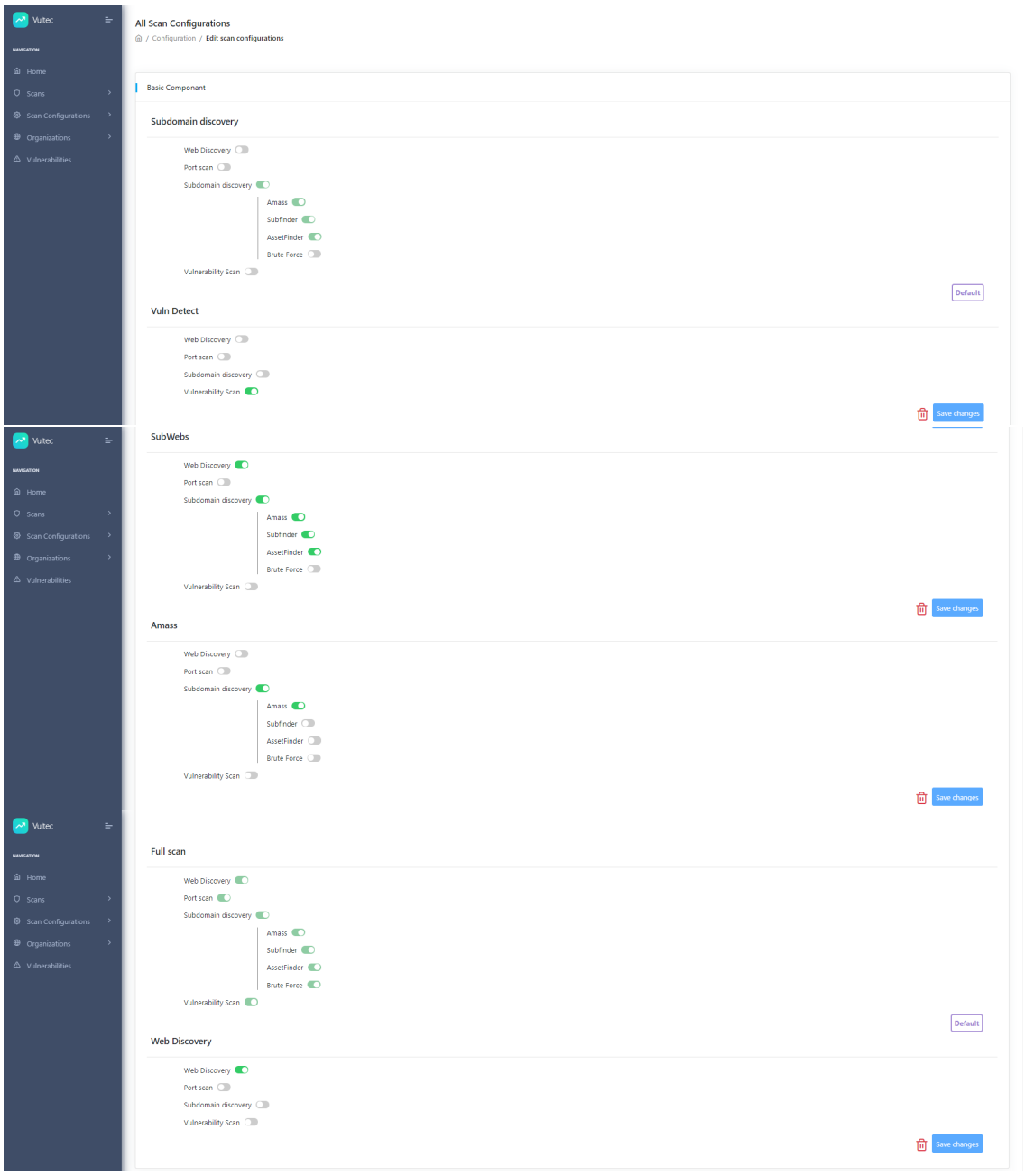


Figura C.4: Página que muestra las configuraciones registradas en vultec.

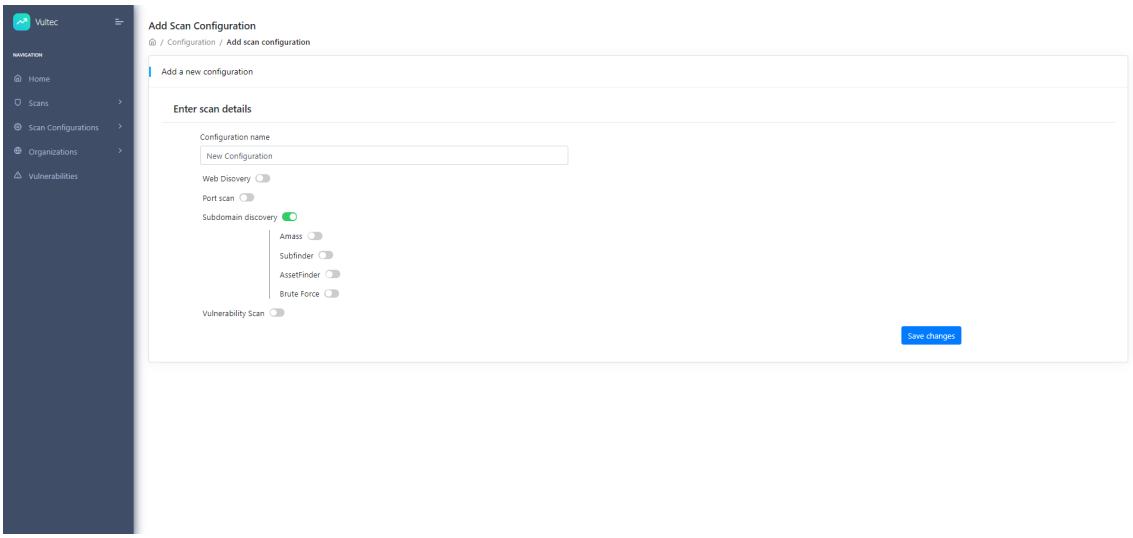


Figura C.5: Página para añadir una nueva configuración a Vultec.

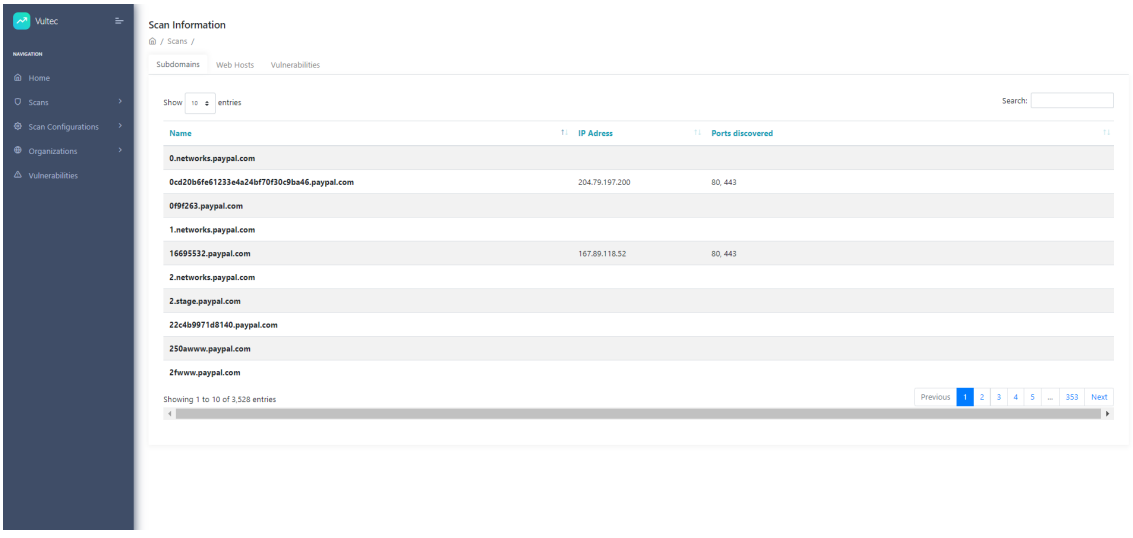


Figura C.6: Tabla que muestra la información relacionada con los subdominios de una organización.

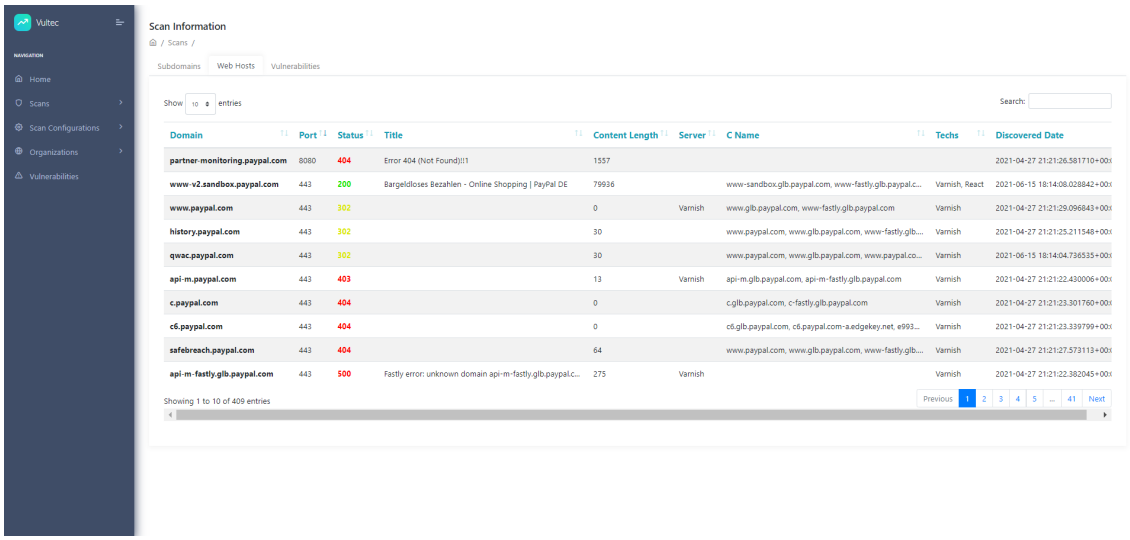


Figura C.7: Tabla que muestra la información relacionada con las webs descubiertas para una organización.

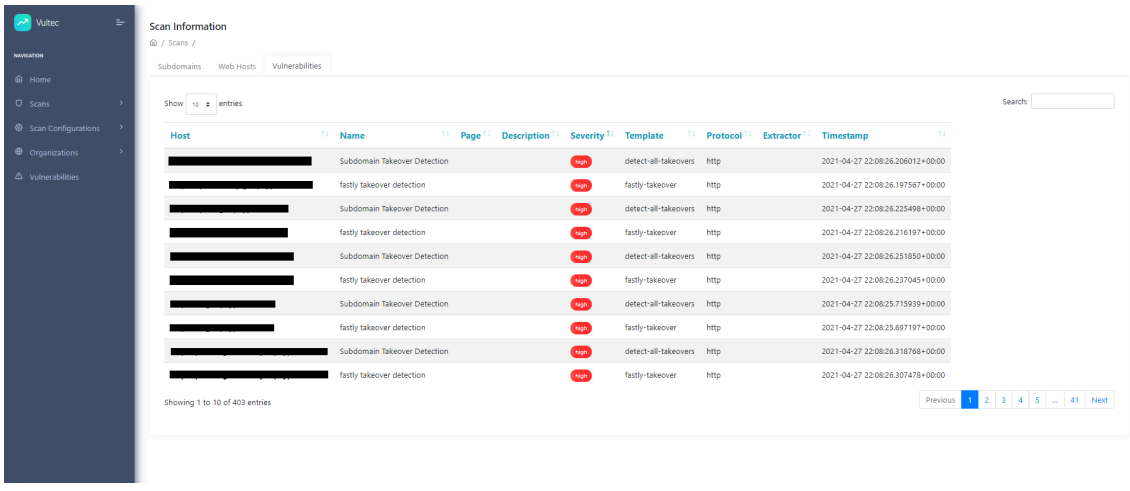


Figura C.8: Tabla que muestra las vulnerabilidades descubiertas en los activos de una organización.

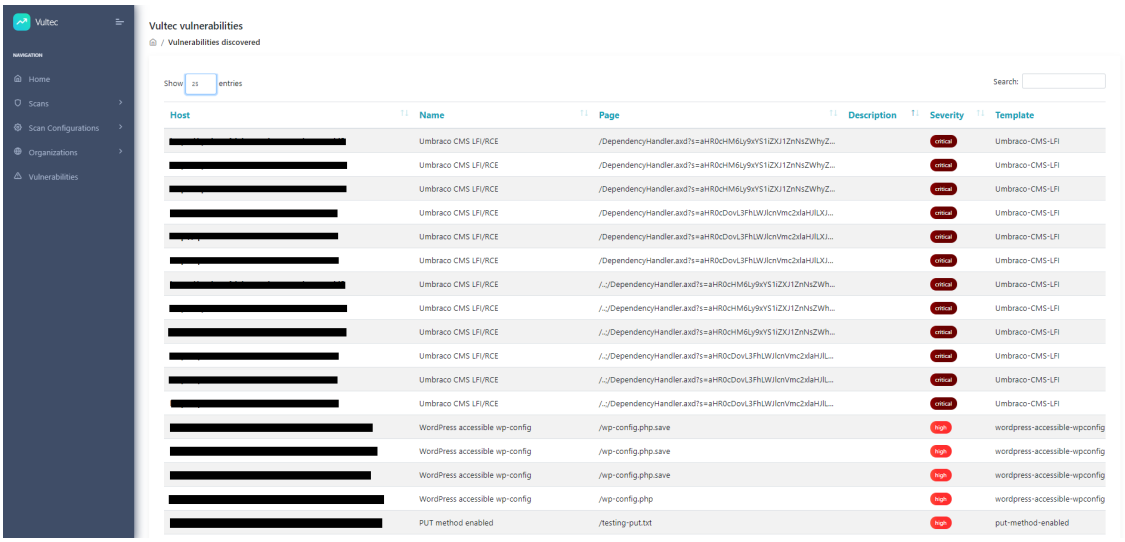


Figura C.9: Tabla que muestra todas las vulnerabilidades descubiertas por Vultec.

D

Instalación de herramienta

En este apéndice se muestran las instalaciones de las herramientas que se han utilizado para el desarrollo de este trabajo de fin de grado.

D.1. Instalación de Go

La mayoría de las siguientes herramientas tienen una programación y sistema de instalación con GoLang, por lo que lo primero será instalar esto para luego poder utilizar el resto. Lo primero será descargar el comprimido del sitio oficial [20], y después ejecutar los siguientes comandos:

```
rm -rf /usr/local/go && tar -C /usr/local -xzf go1.16.5.linux-amd64.tar.gz
export PATH=$PATH:/usr/local/go/bin
```

A partir de este punto, veremos las herramientas ya mencionadas y sus metodos de instalación para su uso directo posteriormente.

D.2. AssetFinder

```
go get -u github.com/tomnomnom/assetfinder
cp /go/bin/assetfinder /bin/
```

D.3. SubFinder

```
go get -v github.com/projectdiscovery/subfinder/v2/cmd/subfinder
cp /go/bin/subfinder /bin/
```

D.4. Amass

```
go get -v github.com/OWASP/Amass cp /go/bin/Amass /bin/
```

D.5. PureDNS

Primero, hay que instalar MassDNS, pues es un requisito ya que utiliza internamente dicha herramienta como ya se ha comentado:

```
git clone https://github.com/blechschmidt/massdns.git

cd massdns

sudo make install
```

Después, se instala PureDNS con los siguientes comandos:

```
go get github.com/d3mondev/puredns/v2

cp /go/bin/puredns /bin/
```

D.6. DNSCewl

```
git clone https://github.com/codingo/DNSCewl.git

cd DNSCewl; cp DNSCewl /bin/
```

D.7. Unfurl

```
go get -u github.com/tomnomnom/unfurl

cp /go/bin/unfurl /bin/
```

D.8. Naabu

```
go get -v github.com/projectdiscovery/naabu/v2/cmd/naabu

cp /go/bin/naabu /bin/
```

D.9. Httpx

```
go get -v github.com/projectdiscovery/httpx/cmd/httpx

cp /go/bin/httpx /bin/
```

D.10. Nuclei

```
go get -v github.com/projectdiscovery/nuclei/v2/cmd/nuclei

cp /go/bin/nuclei /bin/
```

Como se puede comprobar, la instalación es extremadamente sencilla, un motivo mas para haber elegido estas herramientas de forma especifica, por su compatibilidad con Go. Con el segundo comando de cada instalación, copiamos el binario de ejecución en la ruta general, ya que por defecto se incluyen en la ruta de GoLang.

E

Ejecución de herramientas

En este apéndice se mostrará en detalle cada uno de los cuatro scripts principales que mantienen la ejecución de las diferentes tareas en el proceso de un escaneo, para así poder mostrar en detalle los modos de ejecución que se han aplicado en cada una de las herramientas nombradas.

E.1. Enumeración de subdominios

En el caso de la enumeración de subdominios, hay que tener en cuenta que se han paralelizado las 4 siguientes herramientas, en la combinación de uso que especifique el usuario.

```
#!/bin/bash

YELLOW='\033[1;33m'
LIGHT_G='\033[1;32m'
LIGHT_B='\033[1;34m'
RED='\033[0;31m'
NC='\033[0m' # No Color

PATH_RES="/[redacted]/Vultec/scans"

############################## Tool functions ##############################
AssetFinder(){
    echo -e "${YELLOW}Running ${LIGHT_G}AssetFinder${YELLOW}...${NC}"
    assetfinder --subs-only $1 2>/dev/null > $PATH_RES/$domain_selected/subdomains_AssetFinder.txt;
}

SubFinder(){
    echo -e "${YELLOW}Running ${LIGHT_G}SubFinder${YELLOW}...${NC}"
    subfinder -d $1 -recursive -silent -t 200 -o $PATH_RES/$domain_selected/subdomains_SubFinder.txt &> /dev/null;
}

Amass(){
    echo -e "${YELLOW}Running ${LIGHT_G}Amass${YELLOW}...${NC}"
    amass enum -nolocaldb -passive -timeout 10 -silent -min-for-recursive 2 -d $1 -o $PATH_RES/$domain_selected/subdomains_Amass.txt &> /dev/null;
}

BruteForce(){
    echo -e "${YELLOW}Running ${LIGHT_G}Brute Force tools${YELLOW}...${NC}"
    if [ -f $PATH_RES/$domain_selected/subdomains.txt ]; then
        dnscewl -l $PATH_RES/$domain_selected/subdomains.txt -p /[redacted]/wordlists/permutations-prefixes.txt --level 2 --range 20 \
            --sL /[redacted]/wordlists/non-production-hosts.txt --subs | unfurl domains > $PATH_RES/$domain_selected/subdomains_permutations.txt
        sort -u $PATH_RES/$domain_selected/subdomains_permutations.txt -o $PATH_RES/$domain_selected/subdomains_permutations.txt
        puredns resolve $PATH_RES/$domain_selected/subdomains_permutations.txt -r /[redacted]/wordlists/resolvers.txt --wildcard-batch 1000000 \
            -w $PATH_RES/$domain_selected/subdomains_bruteForce.txt &>/dev/null

        if [ `wc -l < $PATH_RES/$domain_selected/subdomains_bruteForce.txt` -lt 3000 ]; then
            cat $PATH_RES/$domain_selected/subdomains_bruteForce.txt | anew $PATH_RES/$domain_selected/subdomains.txt &>/dev/null
        else
            rm -rf $PATH_RES/$domain_selected/subdomains_bruteForce.txt
        fi
    fi
}

###########################################################################
```

Figura E.1: Ejecución de las herramientas de descubrimiento de subdominios

E.2. Escaneo de puertos

```
#!/bin/bash

PATH_RES="/[redacted]/Vultec/scans"
target=$1

naabu -silent -iL $PATH_RES/$target/subdomains.txt -json > $PATH_RES/$target/ports_discovered.txt

```

Figura E.2: Ejecución de Naabu para el escaneo de puertos.

E.3. Descubrimiento de servicios web

```
#!/bin/bash

PATH_RES="/[redacted]/Vultec/scans"
TLD=$1
cat $PATH_RES/$TLD/subdomains.txt | httpx -no-color -status-code -title -content-length -ip -cname -web-server -tech-detect -silent -ports \
    80,443,8000,8080,8983,3000,4443,7000,8090,9090,9443 -timeout 5 -threads 300 -json > $PATH_RES/$TLD/webalives.txt

```

Figura E.3: Ejecución de Httpx para el descubrimiento de servicios web.

E.4. Descubrimiento de vulnerabilidades

```
#!/bin/bash

PATH_RES="/[redacted]/Vultec/scans"
NUCLEI_TMP="/[redacted]/nuclei-templates"
TLD=$1

nuclei -silent -update-templates

if [ ! -f $PATH_RES/$TLD/nuclei_results_draft.txt ]; then
    touch $PATH_RES/$TLD/nuclei_results_draft.txt
fi

nuclei -l $PATH_RES/$TLD/webalives_scan.txt -t $NUCLEI_TMP/cves -t $NUCLEI_TMP/vulnerabilities \
-t $NUCLEI_TMP/exposures -t $NUCLEI_TMP/exposed-panels -t $NUCLEI_TMP/exposed-tokens -t $NUCLEI_TMP/default-logins \
-t $NUCLEI_TMP/misconfiguration -t $NUCLEI_TMP/takeovers -t $NUCLEI_TMP/technologies -t $NUCLEI_TMP/network \
-c 150 -bulk-size 300 -rate-limit 1200 -timeout 10 -retries 2 -r /[redacted]/wordlists/resolvers.txt -json -o $PATH_RES/$TLD/nuclei_results_draft.txt

```

Figura E.4: Ejecución de Nuclei para el descubrimiento de vulnerabilidades.

F

Evidencias y reglas de detección de las vulnerabilidades principales

A continuación se pueden ver las evidencias de las vulnerabilidades que se han mencionado en la sección 4.2, con capturas de pantalla de cada una de ellas. Además, se muestran algunas de las reglas de detección de las vulnerabilidades anteriores. Las reglas que se muestran son las que vienen por defecto en los templates de nuclei.

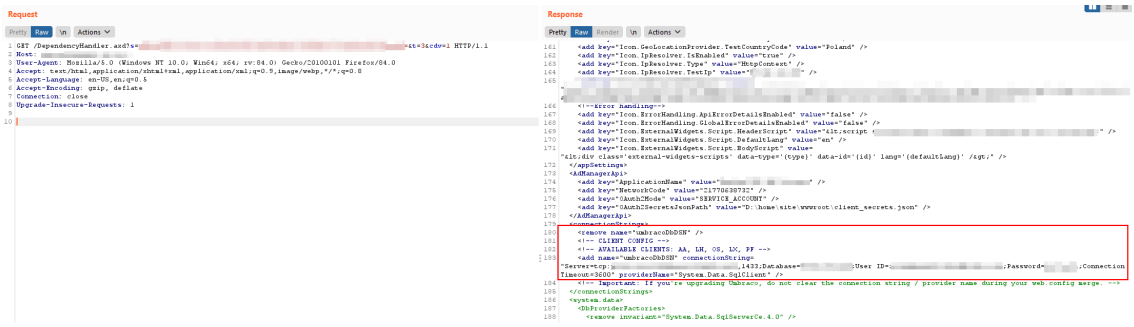


Figura F.1: Evidencias de LFI/RCE

Figura F.2: Evidencias de .env accesible en Larevel

Figura F.3: Evidencias de SSRF

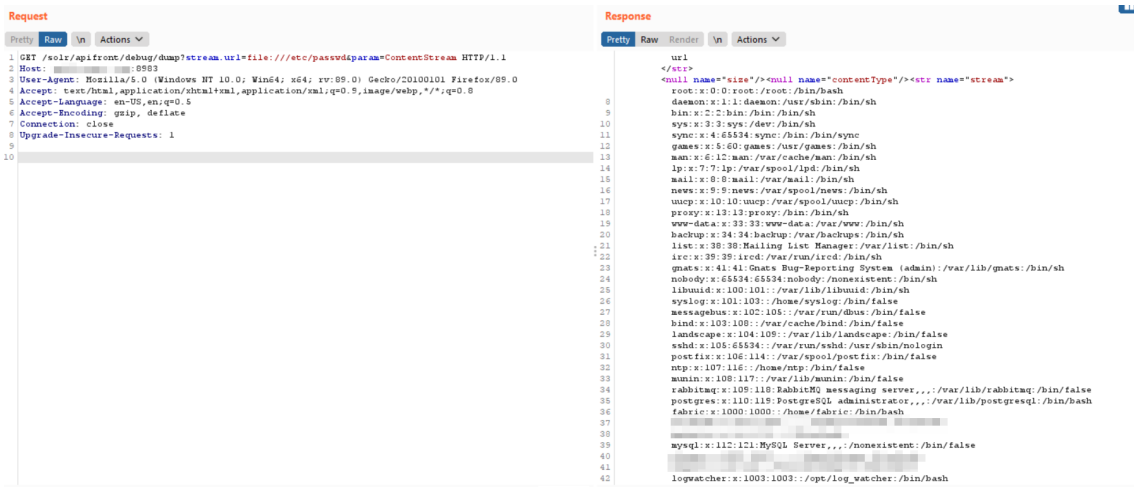


Figura F.4: Evidencias de Apache SOLR

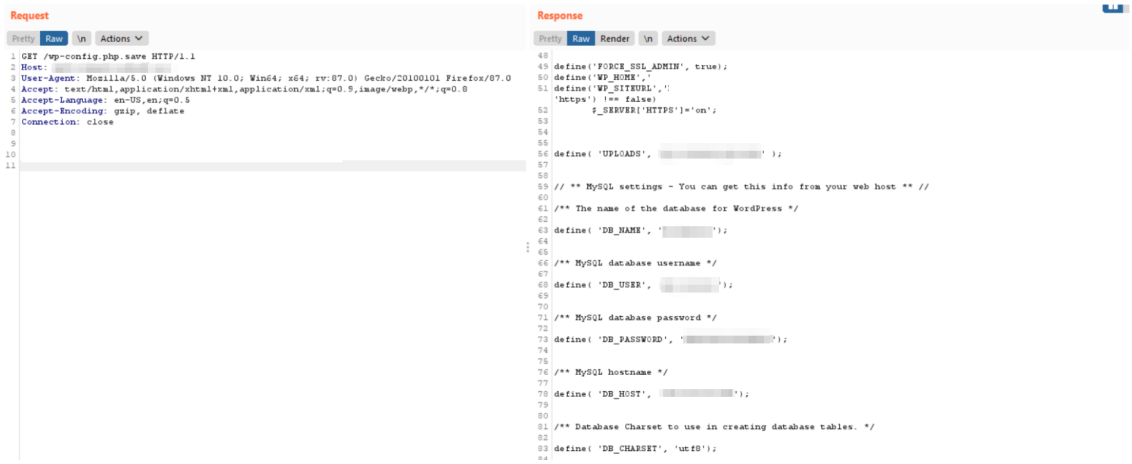


Figura F.5: Evidencias de wp-config accesible

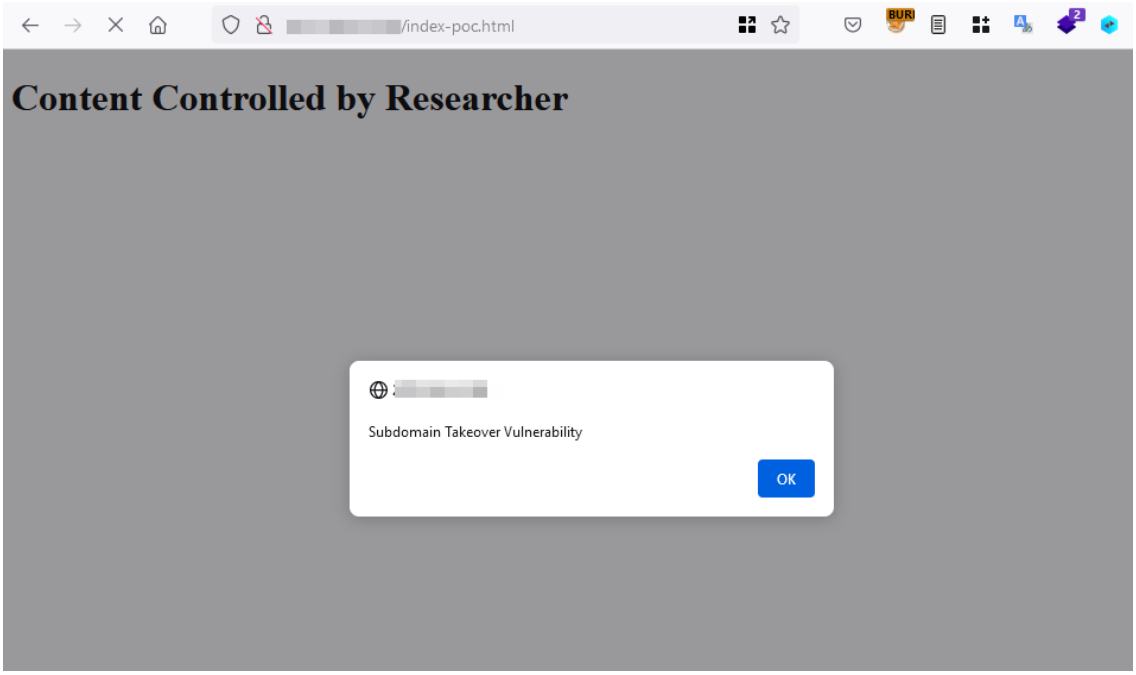


Figura F.6: Evidencias de subdomain takeover

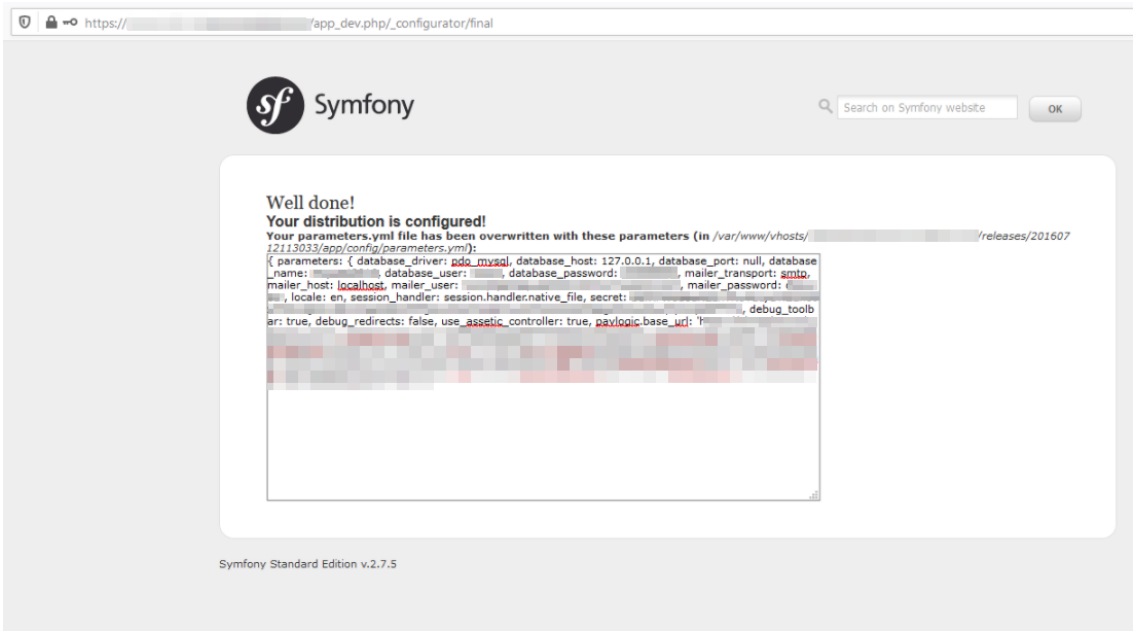


Figura F.7: Evidencias de vulnerabilidad en SymfonyProfiler

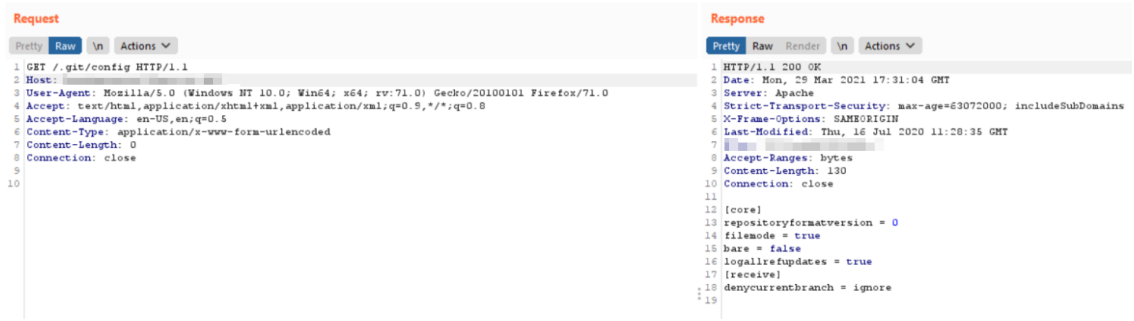


Figura F.8: Evidencias de git config disclosure

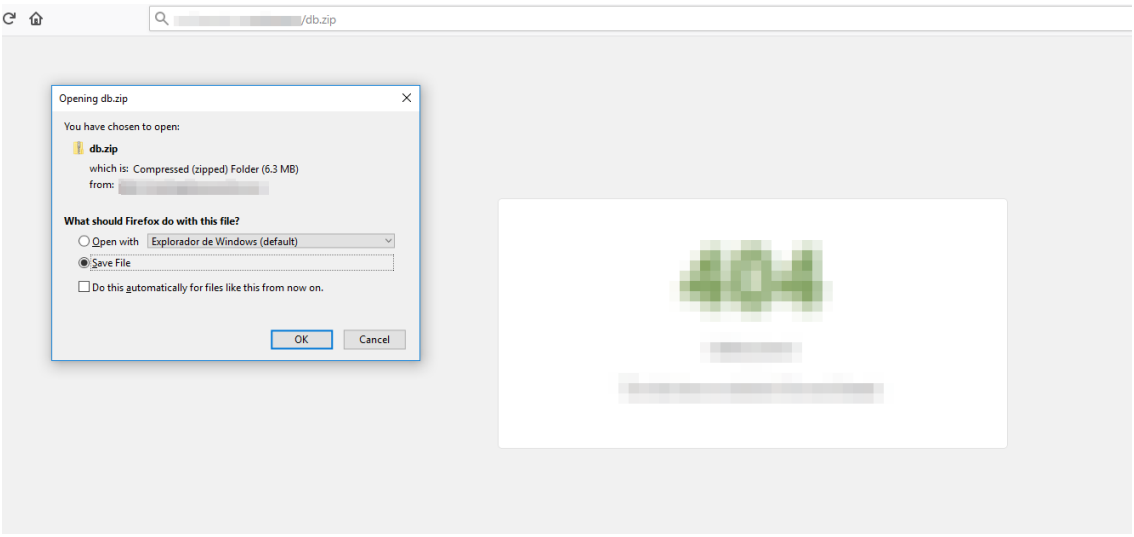


Figura F.9: Evidencias de MySQL Dump Files

```
requests:
  - raw:
      - |
        GET /solr/admin/cores?wt=json HTTP/1.1
        Host: {{Hostname}}
        Accept-Language: en
        Connection: close

      - |
        GET /solr/{{core}}/debug/dump?stream.url=file:///etc/passwd&param=ContentStream HTTP/1.1
        Host: {{Hostname}}
        Accept-Language: en
        Connection: close

    extractors:
      - type: regex
        regex:
          - '"status"\:\{"(.*?)"\:\{"name"'
        name: core
        group: 1
        internal: true

    req-condition: true
    matchers:
      - type: dsl
        dsl:
          - 'status_code_2 == 200'
          - 'regex("root:[x*]:0:0:", body_2)'
        condition: and
```

Figura F.10: Regla de detección de Apache SOLR

```
requests:
  - method: GET
    path:
      - '{{BaseURL}}/wp-config.php'
      - '{{BaseURL}}/.wp-config.php.swp'
      - '{{BaseURL}}/wp-config-sample.php'
      - '{{BaseURL}}/wp-config.inc'
      - '{{BaseURL}}/wp-config.old'
      - '{{BaseURL}}/wp-config.txt'
      - '{{BaseURL}}/wp-config.php.txt'
      - '{{BaseURL}}/wp-config.php.bak'
      - '{{BaseURL}}/wp-config.php.old'
      - '{{BaseURL}}/wp-config.php.dist'
      - '{{BaseURL}}/wp-config.php.inc'
      - '{{BaseURL}}/wp-config.php.swp'
      - '{{BaseURL}}/wp-config.php.html'
      - '{{BaseURL}}/wp-config-backup.txt'
      - '{{BaseURL}}/wp-config.php.save'
      - '{{BaseURL}}/wp-config.php~'
      - '{{BaseURL}}/wp-config.php.orig'
      - '{{BaseURL}}/wp-config.php.original'
      - '{{BaseURL}}/_wpeprivate/config.json'
    matchers-condition: and
    matchers:
      - type: word
        words:
          - DB_NAME
          - WPENGINE_ACCOUNT
        part: body
      - type: status
        status:
          - 200
```

Figura F.11: Regla de detección de Wordpress accessible wp-config

```
requests:
  - method: GET
    path:
      - "{{BaseURL}}/_profiler/empty/search/results?limit=10"
    matchers:
      - type: word
        words:
          - "<title>Symfony Profiler</title>"
          - "symfony/profiler/"
        condition: and
        part: body
```

Figura F.12: Regla de detección de Symfony Profiler information leakage

```
requests:
  - raw:
      - |
        GET /.git/config HTTP/1.1
        Host: {{Hostname}}
        User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0) Gecko/20100101 Firefox/71.0
        Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
        Accept-Language: en-US,en;q=0.5
        Content-Type: application/x-www-form-urlencoded
        Content-Length: 1
        Connection: close

    matchers-condition: and
    matchers:
      - type: word
        words:
          - "[core]"

      - type: dsl
        dsl:
          - 'contains(tolower(body), "<html") == false && contains(tolower(body), "</body>") == false'
```

Figura F.13: Regla de detección de Git Config Disclosure

```
requests:
  - method: GET
    path:
      - "{{BaseURL}}/1.sql"
      - "{{BaseURL}}/backup.sql"
      - "{{BaseURL}}/database.sql"
      - "{{BaseURL}}/data.sql"
      - "{{BaseURL}}/db_backup.sql"
      - "{{BaseURL}}/dbdump.sql"
      - "{{BaseURL}}/db.sql"
      - "{{BaseURL}}/dump.sql"
      - "{{BaseURL}}/{{Hostname}}.sql"
      - "{{BaseURL}}/{{Hostname}}_db.sql"
      - "{{BaseURL}}/localhost.sql"
      - "{{BaseURL}}/mysqldump.sql"
      - "{{BaseURL}}/mysql.sql"
      - "{{BaseURL}}/site.sql"
      - "{{BaseURL}}/sql.sql"
      - "{{BaseURL}}/temp.sql"
      - "{{BaseURL}}/translate.sql"
      - "{{BaseURL}}/users.sql"
      - "{{BaseURL}}/wp-content/uploads/dump.sql"
    headers:
      Range: "bytes=0-3000"

    max-size: 2000 # Size in bytes - Max Size to read from server response
    matchers-condition: and
    matchers:
      - type: regex
        regex:
          - "(?m)(?:DROP|CREATE|(?:UN)?LOCK) TABLE|INSERT INTO"
        part: body

      - type: status
        status:
          - 200
          - 206
        condition: or
```

Figura F.14: Regla de detección de MySql Dump Files


UAM
UNIVERSIDAD AUTONOMA
DE MADRID